

Kai-Min Chung
Yu Sasaki (Eds.)

LNCS 15491

Advances in Cryptology – ASIACRYPT 2024

30th International Conference on the Theory
and Application of Cryptology and Information Security
Kolkata, India, December 9–13, 2024
Proceedings. Part VIII

8
Part VIII



 Springer

Lecture Notes in Computer Science

15491

Founding Editor

Juris Hartmanis


Series Editor

Gerhard Goos, *Karlsruhe Institute of Technology, Karlsruhe, Germany*

Editorial Board Members

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

Wen Gao, *Peking University, Beijing, China*

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

Moti Yung , *Columbia University, New York, 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.


Kai-Min Chung · Yu Sasaki
Editors

Advances in Cryptology – ASIACRYPT 2024

30th International Conference on the Theory
and Application of Cryptology and Information Security
Kolkata, India, December 9–13, 2024
Proceedings, Part VIII

Editors

Kai-Min Chung 
Academia Sinica
Taipei, Taiwan

Yu Sasaki 
NTT Social Informatics Laboratories
Tokyo, Japan

ISSN 0302-9743

ISSN 1611-3349 (electronic)

Lecture Notes in Computer Science

ISBN 978-981-96-0943-7

ISBN 978-981-96-0944-4 (eBook)

<https://doi.org/10.1007/978-981-96-0944-4>

© International Association for Cryptologic Research 2025

This work is subject to copyright. All rights are solely and exclusively licensed 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 Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

If disposing of this product, please recycle the paper.

Preface

The 30th Annual International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt 2024) was held in Kolkata, India, on December 9–13, 2024. The conference covered all technical aspects of cryptology and was sponsored by the International Association for Cryptologic Research (IACR).

We received a record 433 paper submissions for Asiacrypt from around the world. The Program Committee (PC) selected 127 papers for publication in the proceedings of the conference. As in the previous year, the Asiacrypt 2024 program had three tracks.

The two program chairs are greatly indebted to the six area chairs for their great contributions throughout the paper selection process. The area chairs were Siyao Guo for Information-Theoretic and Complexity-Theoretic Cryptography, Bo-Yin Yang for Efficient and Secure Implementations, Goichiro Hanaoka for Public-Key Cryptography Algorithms and Protocols, Arpita Patra for Multi-Party Computation and Zero-Knowledge, Prabhanjan Ananth for Public-Key Primitives with Advanced Functionalities, and Tetsu Iwata for Symmetric-Key Cryptography. The area chairs helped suggest candidates to form a strong program committee, foster and moderate discussions together with the PC members assigned as paper discussion leads to form consensus, suggest decisions on submissions in their areas, and nominate outstanding PC members. We are sincerely grateful for the invaluable contributions of the area chairs.

To review and evaluate the submissions, while keeping the load per PC member manageable, we selected the PC members consisting of 105 leading experts from all over the world, in all six topic areas of cryptology, and we also had approximately 468 external reviewers, whose input was critical to the selection of papers. The review process was conducted using double-blind peer review. The conference operated a two-round review system with a rebuttal phase. This year, we continued the interactive rebuttal from Asiacrypt 2023. After the reviews and first-round discussions, PC members and area chairs selected 264 submissions to proceed to the second round. The remaining 169 papers were rejected, including two desk-rejects. Then, the authors were invited to participate in a two-step interactive rebuttal phase, where the authors needed to submit a rebuttal in five days and then interact with the reviewers to address questions and concerns the following week. We believe the interactive form of the rebuttal encouraged discussions between the authors and the reviewers to clarify the concerns and contributions of the submissions and improved the review process. Then, after several weeks of second-round discussions, the committee selected the final 127 papers to appear in these proceedings. This year, we received seven resubmissions from the revise-and-resubmit experiment from Crypto 2024, of which five were accepted. The nine volumes of the conference proceedings contain the revised versions of the 127 papers that were selected. The final revised versions of papers were not reviewed again and the authors are responsible for their contents.

The PC nominated and voted for three papers to receive the Best Paper Awards. The Best Paper Awards went to Mariya Georgieva Belorgey, Sergiu Carпов, Nicolas Gama,

Sandra Guasch and Dimitar Jetchev for their paper “Revisiting Key Decomposition Techniques for FHE: Simpler, Faster and More Generic”, Xiaoyang Dong, Yingxin Li, Fukang Liu, Siwei Sun and Gaoli Wang for their paper “The First Practical Collision for 31-Step SHA-256”, and Valerio Cini and Hoeteck Wee for their paper “Unbounded ABE for Circuits from LWE, Revisited”. The authors of those three papers were invited to submit extended versions of their papers to the Journal of Cryptology.

The program of Asiacrypt 2024 also featured the 2024 IACR Distinguished Lecture delivered by Paul Kocher and one invited talk, nominated and voted by the PC. The invited speaker had not yet been determined when this preface was written. Following Eurocrypt 2024, we selected seven PC members for the Distinguished PC Members Awards, nominated by the area chairs and program chairs. The Outstanding PC Members Awards went to Sherman S. M. Chow, Elizabeth Crites, Matthias J. Kannwischer, Mustafa Khairallah, Ruben Niederhagen, Maciej Obremski and Keita Xagawa.

Following Crypto 2024, Asiacrypt 2024 included an artifact evaluation process for the first time. Authors of accepted papers were invited to submit associated artifacts, such as software or datasets, for archiving alongside their papers; 14 artifacts were submitted. Rei Ueno was the Artifact Chair and led an artifact evaluation committee of 10 members listed below. In the interactive review process between authors and reviewers, the goal was not just to evaluate artifacts but also to improve them. Artifacts that passed successfully through the artifact review process were publicly archived by the IACR at <https://artifacts.iacr.org/>.

Numerous people contributed to the success of Asiacrypt 2024. We would like to thank all the authors, including those whose submissions were not accepted, for submitting their research results to the conference. We are very grateful to the area chairs, PC members, and external reviewers for contributing their knowledge and expertise, and for the tremendous amount of work that was done with reading papers and contributing to the discussions. We are greatly indebted to Bimal Kumar Roy, the General Chairs, for their efforts in organizing the event, to Kevin McCurley and Kay McKelly for their help with the website and review system, and to Jih-Wei Shih for the assistance with the use of the review system. We thank the Asiacrypt 2024 advisory committee members Bart Preneel, Huaxiong Wang, Bo-Yin Yang, Goichiro Hanaoka, Jian Guo, Ron Steinfeld, and Michel Abdalla for their valuable suggestions. We are also grateful for the helpful advice and organizational material provided to us by Crypto 2024 PC co-chairs Leonid Reyzin and Douglas Stebila, Eurocrypt 2024 PC co-chairs Marc Joye and Gregor Leander, and TCC 2023 chair Hoeteck Wee. We also thank the team at Springer for handling the publication of these conference proceedings.

December 2024

Kai-Min Chung
Yu Sasaki

Pedro Branco	Max Planck Institute for Security and Privacy, Germany
Gaëtan Cassiers	UCLouvain, Belgium
Céline Chevalier	CRED, Université Paris-Panthéon-Assas, and DIENS, France
Avik Chakraborti	Institute for Advancing Intelligence TCG CREST, India
Nishanth Chandran	Microsoft Research India, India
Jie Chen	East China Normal University, China
Yu Long Chen	KU Leuven and National Institute of Standards and Technology, Belgium
Mahdi Cheraghchi	University of Michigan, USA
Nai-Hui Chia	Rice University, USA
Wonseok Choi	Purdue University, USA
Tung Chou	Academia Sinica, Taiwan
Arka Rai Choudhuri	NTT Research, USA
Sherman S. M. Chow	Chinese University of Hong Kong, China
Chitchanok Chuengsatiansup	University of Melbourne, Australia
Michele Ciampi	University of Edinburgh, UK
Valerio Cini	NTT Research, USA
Elizabeth Crites	Web3 Foundation, Switzerland
Nico Döttling	CISPA Helmholtz Center, Germany
Avijit Dutta	Institute for Advancing Intelligence TCG CREST, India
Daniel Escudero	JP Morgan AlgoCRYPT CoE and JP Morgan AI Research, USA
Thomas Espitau	PQShield, France
Jun Furukawa	NEC Corporation, Japan
Rosario Gennaro	CUNY, USA
Junqing Gong	East China Normal University, China
Rishab Goyal	University of Wisconsin-Madison, USA
Julia Hesse	IBM Research Europe, Switzerland
Akinori Hosoyamada	NTT Social Informatics Laboratories, Japan
Michael Hutter	PQShield, Austria
Takanori Isobe	University of Hyogo, Japan
Joseph Jaeger	Georgia Institute of Technology, USA
Matthias J. Kannwischer	Chelpis Quantum Corp, Taiwan
Bhavana Kanukurthi	Indian Institute of Science, India
Shuichi Katsumata	PQShield and AIST, Japan
Jonathan Katz	Google and University of Maryland, USA
Mustafa Khairallah	Lund University, Sweden
Fuyuki Kitagawa	NTT Social Informatics Laboratories, Japan

Karen Klein	ETH Zurich, Switzerland
Mukul Kulkarni	Technology Innovation Institute, United Arab Emirates
Po-Chun Kuo	WisdomRoot Tech, Taiwan
Jooyoung Lee	KAIST, South Korea
Wei-Kai Lin	University of Virginia, USA
Feng-Hao Liu	Washington State University, USA
Jiahui Liu	Massachusetts Institute of Technology, USA
Qipeng Liu	UC San Diego, USA
Shengli Liu	Shanghai Jiao Tong University, China
Chen-Da Liu-Zhang	Lucerne University of Applied Sciences and Arts and Web3 Foundation, Switzerland
Yun Lu	University of Victoria, Canada
Ji Luo	University of Washington, USA
Silvia Mella	Radboud University, Netherlands
Peihan Miao	Brown University, USA
Daniele Micciancio	UCSD, USA
Yusuke Naito	Mitsubishi Electric Corporation, Japan
Khoa Nguyen	University of Wollongong, Australia
Ruben Niederhagen	Academia Sinica, Taiwan and University of Southern Denmark, Denmark
Maciej Obremski	National University of Singapore, Singapore
Miyako Ohkubo	NICT, Japan
Eran Omri	Ariel University, Israel
Jiaxin Pan	University of Kassel, Germany
Anat Paskin-Cherniavsky	Ariel University, Israel
Goutam Paul	Indian Statistical Institute, India
Chris Peikert	University of Michigan, USA
Christophe Petit	University of Birmingham and Université libre de Bruxelles, Belgium
Rachel Player	Royal Holloway University of London, UK
Thomas Prest	PQShield, France
Shahram Rasoolzadeh	Ruhr University Bochum, Germany
Alexander Russell	University of Connecticut, USA
Santanu Sarkar	IIT Madras, India
Sven Schäge	Eindhoven University of Technology, Netherlands
Gregor Seiler	IBM Research Europe, Switzerland
Sruthi Sekar	Indian Institute of Technology, India
Yaobin Shen	Xiamen University, China
Danping Shi	Institute of Information Engineering, Chinese Academy of Sciences, China
Yifan Song	Tsinghua University, China

Katerina Sotiraki	Yale University, USA
Akshayaram Srinivasan	University of Toronto, Canada
Marc Stöttinger	Hochschule RheinMain, Germany
Akira Takahashi	J.P. Morgan AI Research and AlgoCRYPT CoE, USA
Qiang Tang	University of Sydney, Australia
Aishwarya Thiruvengadam	IIT Madras, India
Emmanuel Thomé	Inria Nancy, France
Junichi Tomida	NTT Social Informatics Laboratories, Japan
Monika Trimoska	Eindhoven University of Technology, Netherlands
Huaxiong Wang	Nanyang Technological University, Singapore
Meiqin Wang	Shandong University, China
Qingju Wang	Telecom Paris, Institut Polytechnique de Paris, France
David Wu	UT Austin, USA
Keita Xagawa	Technology Innovation Institute, United Arab Emirates
Chaoping Xing	Shanghai Jiaotong University, China
Shiyuan Xu	University of Hong Kong, China
Anshu Yadav	IST, Austria
Shota Yamada	AIST, Japan
Yu Yu	Shanghai Jiao Tong University, China
Mark Zhandry	NTT Research, USA
Hong-Sheng Zhou	Virginia Commonwealth University, USA

Additional Reviewers

Hugo Aaronson	Jiawei Bao
Damiano Abram	Jyotirmoy Basak
Hamza Abusalah	Nirupam Basak
Abtin Afshar	Gabrielle Beck
Siddharth Agarwal	Hugo Beguinet
Navid Alapati	Amit Behera
Miguel Ambrona	Mihir Bellare
Parisa Amiri Eliasi	Tamar Ben David
Ravi Anand	Aner Moshe Ben Efraim
Saikrishna Badrinarayanan	Fabrice Benhamouda
Chen Bai	Tyler Besselman
David Balbás	Tim Beyne
Brieuc Balon	Rishabh Bhadauria
Gustavo Banegas	Divyanshu Bhardwaj
Laasya Bangalore	Shivam Bhasin

Amit Singh Bhati
Loïc Bidoux
Alexander Bienstock
Jan Bobolz
Alexandra Boldyreva
Maxime Bombar
Nicolas Bon
Carl Bootland
Jonathan Bootle
Giacomo Borin
Cecilia Boschini
Jean-Philippe Bossuat
Mariana Botelho da Gama
Christina Boura
Pierre Briaud
Jeffrey Burdges
Fabio Campos
Yibo Cao
Pedro Capitão
Ignacio Cascudo
David Cash
Wouter Castryck
Anirban Chakrabartha
Debasmita Chakraborty
Suvradip Chakraborty
Kanad Chakravarti
Ayantika Chatterjee
Rohit Chatterjee
Jorge Chavez-Saab
Binyi Chen
Bohang Chen
Long Chen
Mingjie Chen
Shiyao Chen
Xue Chen
Yu-Chi Chen
Chen-Mou Cheng
Jiaqi Cheng
Ashish Choudhury
Miranda Christ
Qiaohan Chu
Eldon Chung
Hao Chung
Léo Colisson
Daniel Collins
Jolijn Cottaar
Murilo Coutinho
Eric Crockett
Bibhas Chandra Das
Nayana Das
Pratish Datta
Alex Davidson
Hannah Davis
Leo de Castro
Luca De Feo
Thomas Decru
Giovanni Deligios
Ning Ding
Fangqi Dong
Minxin Du
Qiuyan Du
Jesko Dujmovic
Moumita Dutta
Pranjal Dutta
Duyen
Marius Eggert
Solane El Hirsch
Andre Esser
Hülya Evkan
Sebastian Faller
Yanchen Fan
Niklas Fassbender
Hanwen Feng
Xiutao Feng
Dario Fiore
Scott Fluhrer
Danilo Francati
Shiuan Fu
Georg Fuchsbauer
Shang Gao
Rachit Garg
Gayathri Garimella
Pierrick Gaudry
François Gérard
Paul Gerhart
Riddhi Ghosal
Shibam Ghosh
Ashrujit Ghoshal
Shane Gibbons
Valerie Gilchrist

Xinxin Gong	Yuval Ishai
Lorenzo Grassi	Ryoma Ito
Scott Griffy	Amit Jana
Chaowen Guan	Ashwin Jha
Aurore Guillevic	Xiaoyu Ji
Sam Gunn	Yanxue Jia
Felix Günther	Mingming Jiang
Kanav Gupta	Lin Jiao
Shreyas Gupta	Haoxiang Jin
Kamil Doruk Gur	Zhengzhong Jin
Jincheol Ha	Chris Jones
Hossein Hadipour	Eliran Kachlon
Tovoheray Hajatiana Randrianarisoa	Giannis Kaklamanis
Shai Halevi	Chethan Kamath
Shuai Han	Soumya Kanti Saha
Tobias Handirk	Sabyasachi Karati
Yonglin Hao	Harish Karthikeyan
Zihan Hao	Andes Y. L. Kei
Keisuke Hara	Jean Kieffer
Keitaro Hashimoto	Jiseung Kim
Aditya Hegde	Seongkwang Kim
Andreas Hellenbrand	Sebastian Kolby
Paul Hermouet	Sreehari Kollath
Minki Hhan	Dimitris Kolonelos
Hilder Lima	Venkata Koppula
Taiga Hiroka	Abhiram Kothapalli
Ryo Hiromasa	Stanislav Kruglik
Viet Tung Hoang	Anup Kumar Kundu
Charlotte Hoffmann	Péter Kutas
Clément Hoffmann	Norman Lahr
Man Hou Hong	Qiqi Lai
Wei-Chih Hong	Yi-Fu Lai
Alexander Hoover	Abel Laval
Fumitaka Hoshino	Guirec Lebrun
Patrick Hough	Byeonghak Lee
Yao-Ching Hsieh	Changmin Lee
Chengcong Hu	Hyung Tae Lee
David Hu	Joohee Lee
Kai Hu	Keewoo Lee
Zihan Hu	Yeongmin Lee
Hai Huang	Yongwoo Lee
Mi-Ying Huang	Andrea Lesavourey
Yu-Hsuan Huang	Baiyu Li
Zhicong Huang	Jiangtao Li
Shih-Han Hung	Jianqiang Li

Junru Li
Liran Li
Minzhang Li
Shun Li
Songsong Li
Weihan Li
Wenzhong Li
Yamin Li
Yanan Li
Yu Li
Yun Li
Zeyong Li
Zhe Li
Chuanwei Lin
Fuchun Lin
Yao-Ting Lin
Yunhao Ling
Eik List
Fengrun Liu
Fukang Liu
Hanlin Liu
Hongqing Liu
Rui Liu
Tianren Liu
Xiang Liu
Xiangyu Liu
Zeyu Liu
Paul Lou
George Lu
Zhenghao Lu
Ting-Gian Lua
You Lyu
Jack P. K. Ma
Yiping Ma
Varun Madathil
Lorenzo Magliocco
Avishek Majumder
Nikolaos Makriyannis
Varun Maram
Chloe Martindale
Elisaweta Masserova
Jake Massimo
Loïc Masure
Takahiro Matsuda
Christian Matt
Subhra Mazumdar
Nikolas Melissaris
Michael Meyer
Ankit Kumar Misra
Anuja Modi
Deep Inder Mohan
Charles Momin
Johannes Mono
Hart Montgomery
Ethan Mook
Thorben Moos
Tomoyuki Morimae
Hiraku Morita
Tomoki Moriya
Aditya Morolia
Christian Mouchet
Nicky Mouha
Tamer Mour
Changrui Mu
Arindam Mukherjee
Pratyay Mukherjee
Anne Müller
Alice Murphy
Shyam Murthy
Kohei Nakagawa
Barak Nehoran
Patrick Neumann
Lucien K. L. Ng
Duy Nguyen
Ky Nguyen
Olga Nissenbaum
Anca Nitulescu
Julian Nowakowski
Frederique Oggier
Jean-Baptiste Orfila
Emmanuela Orsini
Tapas Pal
Ying-yu Pan
Roberto Parisella
Aditi Partap
Alain Passelègue
Alice Pellet-Mary
Zachary Pepin
Octavio Perez Kempner
Edoardo Perichetti

Léo Perrin
Naty Peter
Richard Petri
Rafael del Pino
Federico Pintore
Erik Pohle
Simon Pohmann
Guru Vamsi Policharla
Daniel Pollman
Yuriy Polyakov
Alexander Poremba
Eamonn Postlethwaite
Sihang Pu
Luowen Qian
Tian Qiu
Rajeev Raghunath
Srinivasan Raghuraman
Mostafizar Rahman
Mahesh Rajasree
Somindu Chaya Ramanna
Simon Rastikian
Anik Raychaudhuri
Martin Rehberg
Michael Reichle
Krijn Reijnders
Doreen Riepel
Guilherme Rito
Matthieu Rivain
Bhaskar Roberts
Marc Roeschlin
Michael Rosenberg
Paul Rösler
Arnab Roy
Lawrence Roy
Luigi Russo
Keegan Ryan
Markku-Juhani Saarinen
Éric Sageloli
Dhiman Saha
Sayandeep Saha
Yusuke Sakai
Kosei Sakamoto
Subhabrata Samajder
Simona Samardjiska
Maria Corte-Real Santos
Sina Schaeffler
André Schrottenloher
Jacob Schuldt
Mark Schultz
Mahdi Sedaghat
Jae Hong Seo
Yannick Seurin
Aein Shahmirzadi
Girisha Shankar
Yixin Shen
Rentaro Shiba
Ardeshir Shojaeinasab
Jun Jie Sim
Mark Simkin
Jaspal Singh
Benjamin Smith
Yongha Son
Fang Song
Yongsoo Song
Pratik Soni
Pierre-Jean Spaenlehauer
Matthias Johann Steiner
Lukas Stennes
Roy Stracovsky
Takeshi Sugawara
Adam Suhl
Siwei Sun
Elias Suvanto
Koutarou Suzuki
Erkan Tairi
Atsushi Takayasu
Kaoru Takemure
Abdullah Talayhan
Quan Quan Tan
Gang Tang
Khai Hanh Tang
Tianxin Tang
Yi Tang
Stefano Tessaro
Sri AravindaKrishnan Thyagarajan
Yan Bo Ti
Jean-Pierre Tillich
Toi Tomita
Aleksei Udovenko
Arunachalaeswaran V.

Aron van Baarsen	Kyosuke Yamashita
Wessel van Woerden	Jiayun Yan
Michiel Verbauwhede	Yingfei Yan
Corentin Verhamme	Qianqian Yang
Quoc-Huy Vu	Rupeng Yang
Benedikt Wagner	Xinrui Yang
Julian Wälde	Yibin Yang
Hendrik Waldner	Zhaomin Yang
Judy Walker	Yizhou Yao
Alexandre Wallet	Kevin Yeo
Han Wang	Eylon Yogev
Haoyang Wang	Yusuke Yoshida
Jiabo Wang	Aaram Yun
Jiafan Wang	Gabriel Zaid
Liping Wang	Riccardo Zanotto
Mingyuan Wang	Shang Zehua
Peng Wang	Hadas Zeilberger
Weihao Wang	Runzhi Zeng
Yunhao Wang	Bin Zhang
Zhedong Wang	Cong Zhang
Yohei Watanabe	Liu Zhang
Chenkai Weng	Tianwei Zhang
Andreas Weninger	Tianyu Zhang
Stella Wohnig	Xiangyang Zhang
Harry W. H. Wong	Yijian Zhang
Ivy K. Y. Woo	Yinuo Zhang
Tiger Wu	Yuxin Zhang
Yu Xia	Chang-an Zhao
Zejun Xiang	Tianyu Zhao
Yuting Xiao	Yu Zhou
Ning Xie	Yunxiao Zhou
Zhiye Xie	Zhelei Zhou
Lei Xu	Zibo Zhou
Yanhong Xu	Chenzhi Zhu
Haiyang Xue	Ziqi Zhu
Aayush Yadav	Cong Zuo
Saikumar Yadugiri	

Artifact Chair

Rei Ueno

Kyoto University, Japan

Artifact Evaluation Committee

Julien Béguinot	LTCl, Télécom Paris, Institut Polytechnique de Paris, France
Aron Gohr	Independent Researcher
Hosein Hadipour	Graz University of Technology, Austria
Akira Ito	NTT Social Informatics Laboratories, Japan
Haruto Kimura	University of Melbourne, Australia and Waseda University, Japan
Kotaro Matsuoka	Kyoto University, Japan
Florian Mendel	Infineon Technologies, Germany
Hiraku Morita	Aarhus University, University of Copenhagen, Denmark
Prasanna Ravi	Nanyang Technological University, Singapore
Élise Tasso	Tohoku University, Japan

Contents – Part VIII

Cryptanalysis on Public-Key Schemes

Attacking ECDSA with Nonce Leakage by Lattice Sieving: Bridging the Gap with Fourier Analysis-Based Attacks	3
<i>Yiming Gao, Jinghui Wang, Honggang Hu, and Binang He</i>	
Don't Use it Twice! Solving Relaxed Linear Equivalence Problems	35
<i>Alessandro Budroni, Jesús-Javier Chi-Domínguez, Giuseppe D'Alconzo, Antonio J. Di Scala, and Mukul Kulkarni</i>	
Rare Structures in Tensor Graphs: Bermuda Triangles for Cryptosystems Based on the Tensor Isomorphism Problem	66
<i>Lars Ran and Simona Samardjiska</i>	

Fault Attacks and Side-Channel Analysis

It's a Kind of Magic: A Novel Conditional GAN Framework for Efficient Profiling Side-Channel Analysis	99
<i>Sengim Karayalçın, Marina Krček, Lichao Wu, Stjepan Picek, and Guilherme Perin</i>	
ZKFault: Fault Attack Analysis on Zero-Knowledge Based Post-quantum Digital Signature Schemes	132
<i>Puja Mondal, Supriya Adhikary, Suparna Kundu, and Angshuman Karmakar</i>	
More Vulnerabilities of Linear Structure Sbox-Based Ciphers Reveal Their Inability to Resist DFA	168
<i>Amit Jana, Anup Kumar Kundu, and Goutam Paul</i>	

Cryptanalysis on Various Problems

Hard-Label Cryptanalytic Extraction of Neural Network Models	207
<i>Yi Chen, Xiaoyang Dong, Jian Guo, Yantian Shen, Anyu Wang, and Xiaoyun Wang</i>	
Analysis on Sliced Garbling via Algebraic Approach	237
<i>Taechan Kim</i>	

Revisiting OKVS-Based OPRF and PSI: Cryptanalysis and Better Construction	266
<i>Kyoohyung Han, Seongkwang Kim, Byeonghak Lee, and Yongha Son</i>	
Quantum Cryptanalysis	
Reducing the Number of Qubits in Quantum Information Set Decoding	299
<i>Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher</i>	
On the Semidirect Discrete Logarithm Problem in Finite Groups	330
<i>Christopher Battarbee, Giacomo Borin, Julian Brough, Ryann Cartor, Tobias Hemmert, Nadia Heninger, David Jao, Delaram Kahrobaei, Laura Maddison, Edoardo Persichetti, Angela Robinson, Daniel Smith-Tone, and Rainer Steinwandt</i>	
Quantum Circuits of AES with a Low-Depth Linear Layer and a New Structure	358
<i>Haotian Shi and Xiutao Feng</i>	
Quantum Algorithms for Fast Correlation Attacks on LFSR-Based Stream Ciphers	396
<i>Akinori Hosoyamada</i>	
Author Index	431

Cryptanalysis on Public-Key Schemes



Attacking ECDSA with Nonce Leakage by Lattice Sieving: Bridging the Gap with Fourier Analysis-Based Attacks

Yiming Gao¹, Jinghui Wang¹, Honggang Hu^{1,2(✉)}, and Binang He¹

¹ School of Cyber Science and Technology, University of Science and Technology of China, Hefei 230027, China

{qw1234567, liqing21, hebinang}@mail.ustc.edu.cn

² Hefei National Laboratory, Hefei 230088, China

hghu2005@ustc.edu.cn

Abstract. The Hidden Number Problem (HNP) has found extensive applications in side-channel attacks against cryptographic schemes, such as ECDSA and Diffie-Hellman. There are two primary algorithmic approaches to solving the HNP: lattice-based attacks and Fourier analysis-based attacks. Lattice-based attacks exhibit better efficiency and require fewer samples when sufficiently long substrings of the nonces are known. However, they face significant challenges when only a small fraction of the nonce is leaked, such as 1-bit leakage, and their performance degrades in the presence of errors.

In this paper, we address an open question by introducing an algorithmic tradeoff that significantly bridges the gap between these two approaches. By introducing a parameter x to modify Albrecht and Heninger's lattice, the lattice dimension is reduced by approximately $(\log_2 x)/l$, where l represents the number of leaked bits. We present a series of new methods, including the interval reduction algorithm, several predicates, and the pre-screening technique. Furthermore, we extend our algorithms to solve the HNP with erroneous input. Our attack outperforms existing state-of-the-art lattice-based attacks against ECDSA. We obtain several records including 1-bit and less than 1-bit leakage on a 160-bit curve, while the best previous lattice-based attack for 1-bit leakage was conducted only on a 112-bit curve.

Keywords: ECDSA · Hidden Number Problem · Lattice Sieving · Lattice-based Attacks

1 Introduction

The Hidden Number Problem (HNP) was originally proposed by Boneh and Venkatesan as a number theoretic problem to investigate the bit security of the Diffie-Hellman key exchange scheme [12]. Their work was subsequently extended

Yiming Gao and Jinghui Wang are the co-first authors of this work.

© International Association for Cryptologic Research 2025

K.-M. Chung and Y. Sasaki (Eds.): ASIACRYPT 2024, LNCS 15491, pp. 3–34, 2025.

https://doi.org/10.1007/978-981-96-0944-4_1

by Nguyen and Shparlinski to analyze the security of ECDSA with partial known nonce leakage [29]. In the scenario that an attacker can obtain some information about the nonce used in each signature generation of ECDSA, it is possible to recover the secret key by solving the corresponding HNP instance. Currently, there are two types of attacks for solving the HNP: Fourier analysis-based attacks and lattice-based attacks.

The foundational principles of Fourier analysis-based attacks were initially introduced by Bleichenbacher [10], providing the basis for subsequent research [5, 6, 14, 35]. These attacks have been considered to be more tractable to break HNP instances with limited known bits and even with errors. However, they demand a substantial number of samples and exhibit a high computational overhead [5, 6, 11]. The latest advance is obtained by Aranha et al., who successfully broke 192-bit ECDSA with less than 1-bit leakage [6].

In lattice-based attacks, the HNP is transformed into the Bounded Distance Decoding (BDD) Problem, which is a variant of the Closest Vector Problem (CVP). Using Kannan’s embedding [23], it can be further transformed into the Unique Shortest Vector Problem (uSVP). The success of lattice-based attacks highly depends on whether the target vector corresponding to the secret is sufficiently short in the lattice. It is believed that the lattice-based attacks would become ineffective when only a small fraction of the nonce is revealed, particularly in the case of 1-bit leakage. Aranha et al. emphasized that exploiting a 1-bit nonce leakage to attack ECDSA is infeasible due to the underlying structure of the HNP lattices [5]. Additionally, lattice-based attacks have been considered to behave very poorly with noisy data, which poses constraints on practical side-channel attacks.

At EUROCRYPT 2021, Albrecht and Heninger extended the applicability of lattice-based attacks with their Sieving with Predicate (Sieve-Pred) algorithm [3] and the state-of-the-art lattice sieving library G6K [2]. The Sieve-Pred algorithm no longer treats sieving algorithms as black boxes for SVP. Instead, it uses a predicate to check all the vectors in the database output by sieving algorithms. The predicate they used involves scalar multiplication on the elliptic curve, which is a nonlinear operation and results in significant overhead. The nonlinear predicate was improved to a linear predicate by Xu et al. [38].

Lattice-based attacks and Fourier analysis-based attacks have their unique characteristics. Current lattice-based attacks are known for their minimal sample requirements and efficient processing. However, they are considered to be infeasible when dealing with challenging HNP instances. In contrast, Fourier analysis-based attacks can handle more difficult instances such as 1-bit leakage on a 192-bit curve [6], but demand a significantly larger sample size and computational time. This raises open questions [20]: *Can lattice-based attacks be enhanced by utilizing more samples? Is there a smooth tradeoff that can be characterized between these two types of algorithms?*

1.1 Contributions

In this work, we enhance the lattice-based attacks by utilizing more samples, significantly bridging the gap between lattice-based attacks and Fourier analysis-based attacks. The main idea of our attack is using more samples to obtain the specific ones employed in the lattice construction. This allows us to increase the lattice determinant while keeping the norm of the target vector roughly unchanged. Consequently, it results in a reduction of lattice dimension while still satisfying the enabling condition for the attack.

In practice, we successfully address the case of 1-bit leakage on a 160-bit curve, surpassing the best previous lattice-based attack for 1-bit leakage, which was only conducted on a 112-bit curve [38]. Moreover, despite the belief that lattice-based attacks are ineffective for erroneous input [6], we demonstrate the effectiveness of our new attack in handling erroneous input, and successfully break the 160-bit ECDSA with less than 1-bit leakage. Our main contributions are detailed as follows.

Improved Algorithms for Solving the HNP. Firstly, we propose a new lattice construction that introduces a parameter x to trade off the lattice dimension. Our modification is based on Albrecht and Heninger’s lattice [3], where the hidden number α is transformed into k'_0 using the elimination method and the recentering technique.

In our construction, we employ a decomposition technique in which k'_0 is decomposed as $x \cdot \alpha_0 + \alpha_1$, with the condition that $|\alpha_1| \leq x/2$. The target lattice vector contains the information of α_0 , which is expected to be included in the database output by sieving algorithms.

Compared to Albrecht and Heninger’s lattice [3], our construction can offer a dimension reduction of approximately $(\log_2 x)/l$, where l represents the number of leaked bits. The reduction in lattice dimension leads to a significant efficiency advantage because sieving algorithms have exponential complexity. As a tradeoff, our algorithm needs a larger number of samples to select the specific ones used in the lattice construction. Moreover, we prove the existence of a constant $c > 0$ which serves as a lower bound for the success probability of our algorithm. This is a new theoretical finding not reported in the literature.

Secondly, in order to determine the unique hidden number, we propose an improved linear predicate that makes use of the linear constraints from $2 \log_2 q$ HNP samples. Our predicate demonstrates superior efficiency compared to the non-linear predicate proposed by Albrecht and Heninger [3], which involves time-consuming scalar multiplication over elliptic curves. Our predicate also outperforms the linear predicate used by Xu et al. [38]. Their predicate requires knowledge of all elements of the candidate vector, whereas ours only requires the last two elements. Furthermore, we identify an issue with Xu et al.’s predicate that may cause it to return true for some incorrect candidates. We provide a corrected version of their predicate for a fair comparison and demonstrate that our approach achieves superior performance while maintaining the desired functionality.

Thirdly, a predicate for the decomposition technique is proposed. We design an interval reduction algorithm with expected time complexity $O(\log^2 x)$ to

recover the remaining part α_1 , instead of an exhaustive search over the range $[-x/2, x/2]$. Moreover, we also present a pre-screening technique to pre-select candidates. This technique can effectively eliminate most incorrect candidates before checking the predicate.

Modified Algorithms for Handling Errors. We define HNP with erroneous input to handle practical scenarios in side-channel attacks where errors may exist in the leaked nonce. We demonstrate the effectiveness of our lattice construction for solving this problem, and provide the estimation for the minimum lattice dimension. In this case, our construction offers a greater reduction in lattice dimension of more than $\log_2 x/l$. Furthermore, our new algorithms and techniques for solving the HNP are extended to address the case of erroneous input.

New Records of Lattice-Based Attacks Against ECDSA. We carry out experiments on lattice-based attacks against ECDSA with nonce leakage. Our attack demonstrates a significant efficiency advantage over previous works [3, 34, 38]. The most notable achievement is the successful key recovery for the ECDSA instance with 1-bit leakage on a 160-bit curve, which is considered to be extremely difficult by previous lattice-based approaches. Moreover, we also successfully conduct attacks for the case of less than 1-bit leakage on various elliptic curves, including a 160-bit curve. Our source code is publicly available on GitHub¹.

1.2 Comparison with Related Work

In Table 1, our work is compared with previous records of lattice-based attacks. Several new records are listed, including 1-bit and less than 1-bit leakage on a 160-bit curve. To the best of our knowledge, we carry out the first lattice-based attack against ECDSA with less than 1-bit leakage (the leaked 1-bit of the nonce is exact with a probability of less than 1). For the case of 4-bit leakage, we also make significant progress. While the previous record for 4-bit leakage was achieved on a 384-bit curve [3, 34], we extend the attack to a 512-bit curve.

Among lattice-based attacks, the currently best one is the Sieve-Pred algorithm in [3]. The efficiency of the predicate is essential, as it is used to check the vectors in the database generated by sieving algorithms. Our linear predicate outperforms both predicates proposed in [3] and [38]. Our attack demonstrates superior efficiency when targeting various ECDSA instances. For example, we break the 112-bit ECDSA with 1-bit leakage in 6 min which is approximately 43 times faster than the currently fastest attack in [38].

Our work can also be viewed as a comprehensive extension of the work conducted by Sun et al. [34], who recognized the connection between Fourier-based attacks and lattice-based attacks, and proposed a general framework to enhance lattice attacks with more samples. However, they did not consider reducing the

¹ <https://github.com/JinghuiWW/ecdsa-leakage-attack>.

Table 1. Lattice-based attacks against ECDSA with nonce leakage

Modulus	Nonce leakage				
	4-bit	3-bit	2-bit	1-bit	<1-bit
112-bit	-	-	-	[38], Ours (faster)	Ours
128-bit	-	-	-	Ours	Ours
160-bit	-	[29]	[3, 34]	Ours	Ours
256-bit	[3]	[3, 34]	[38], Ours (faster)	-	-
384-bit	[3, 34] [38], Ours (faster)		-	-	-
512-bit	Ours	-	-	-	-

lattice dimension, nor did they utilize sieving algorithms to handle more challenging cases, such as 256-bit ECDSA with 2-bit leakage or 1-bit leakage case.

Compared with their work, our algorithm requires significantly fewer samples and achieves higher success rates. For instance, when targeting 160-bit ECDSA with 2-bit leakage, their algorithm required approximately 2^{27} samples and achieved a success rate of 15%, while our algorithm only needs 411 samples with a success rate of approximately 100%. Moreover, according to their estimation, the time complexity of their algorithm is 2^{110} BKZ-30 operations for 160-bit ECDSA with 1-bit leakage, which is impractical. However, we break this instance in only 13.7 h using approximately 2^{25} ECDSA samples.

Our work shares some similarities with Fourier analysis-based attacks. However, except for very difficult cases, our attack works more efficiently with far fewer samples. Another advantage of our approach is the capability to recover the entire secret key all at once, while Fourier analysis-based attacks can only recover a few bits of the secret key in a single execution.

2 Preliminaries

Let \mathbb{N}^+ be the set $\{1, 2, 3, \dots\}$. The logarithm with base two is denoted as $\log(\cdot)$, and the Euclidean norm is denoted as $\|\cdot\|$. For any integer z , the unique integer x satisfying $0 \leq x < q$ and $x \equiv z \pmod{q}$ is denoted by $|z|_q$. A function $f(k)$ is called $\text{negl}(k)$ if for every positive polynomial function $P(k)$, there exists an integer N_P ≥ 0 such that for all $k > N_P$, $|f(k)| < 1/P(k)$.

2.1 Lattices and Hard Problems

Given a matrix $\mathbf{B} = (\mathbf{b}_0, \dots, \mathbf{b}_{d-1})^T \in \mathbb{R}^{d \times d}$ with linearly independent rows, the lattice generated by the basis \mathbf{B} is defined as $\mathcal{L}(\mathbf{B}) := \{\sum_i^d x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$. We define π_i as the projections orthogonal to the span of $\mathbf{b}_0, \dots, \mathbf{b}_{i-1}$, and the Gram-Schmidt orthogonalisation as $\mathbf{B}^* = (\mathbf{b}_0^*, \dots, \mathbf{b}_{d-1}^*)$, where $\mathbf{b}_i^* := \pi_i(\mathbf{b}_i)$. For any $0 \leq l < r \leq d$, the projected sublattice $\mathcal{L}_{[l:r]}$ is defined as the lattice with basis $\mathbf{B}_{[l:r]} := (\pi_l(\mathbf{b}_l), \dots, \pi_l(\mathbf{b}_{r-1}))$.

Let $\lambda_i(\mathcal{L})$ be the radius of the smallest ball centred at the origin containing at least i linearly independent lattice vectors. Then $\lambda_1(\mathcal{L})$ is the Euclidean norm of the shortest non-zero vector in lattice \mathcal{L} .

The Gaussian heuristic predicts the number of lattice points inside a measurable body $\mathcal{B} \subset \mathbb{R}^n$, and it tells us that the number $|\mathcal{L} \cap \mathcal{B}|$ of lattice points inside \mathcal{B} is approximately equal to $\text{Vol}(\mathcal{B})/\text{Vol}(\mathcal{L})$. Applying it to the Euclidean d -ball, the prediction of $\lambda_1(\mathcal{L})$ can be obtained.

Definition 1 (Gaussian Heuristic (GH)). We denote by $\text{GH}(\mathcal{L})$ the expected first minimum of a lattice \mathcal{L} . For a full rank lattice $\mathcal{L} \subset \mathbb{R}^d$, it is given by

$$\text{GH}(\mathcal{L}) = \frac{\Gamma(1 + \frac{d}{2})^{1/d}}{\sqrt{\pi}} \cdot \text{Vol}(\mathcal{L})^{1/d} \approx \sqrt{\frac{d}{2\pi e}} \cdot \text{Vol}(\mathcal{L})^{1/d}.$$

The final step above is obtained from Stirling's formula, and we utilize this asymptotic estimation in our theoretical analysis. In practical attacks and calculations, we directly compute the value of the Gamma function.

Albrecht and Heninger formalized two lattice problems augmented with a predicate [3], which are defined as follows:

Definition 2 (α -Bounded Distance Decoding with Predicate ($\text{BDD}_{\alpha, f(\cdot)}$)). Given a lattice basis \mathbf{B} , a vector \mathbf{t} and a parameter $\alpha > 0$ such that the Euclidean distance $\text{dist}(\mathbf{t}, \mathbf{B}) < \alpha \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$, find the lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ satisfying $f(\mathbf{v} - \mathbf{t}) = 1$ that is closest to \mathbf{t} .

Definition 3 (unique Shortest Vector Problem with Predicate ($\text{uSVP}_{f(\cdot)}$)). Given a lattice basis \mathbf{B} and a predicate $f(\cdot)$, find the shortest non-zero vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ that satisfies $f(\mathbf{v}) = 1$.

$\text{BDD}_{\alpha, f(\cdot)}$ can be solved using a $\text{uSVP}_{f(\cdot)}$ oracle due to Kannan embedding technique, by constructing the lattice

$$\mathbf{C} = \begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{t} & \tau \end{bmatrix},$$

where τ is the embedding factor. If \mathbf{v} is the closest vector to \mathbf{t} , then the short vector $(\mathbf{t} - \mathbf{v}, \tau)$ is contained in the lattice $\mathcal{L}(\mathbf{C})$.

2.2 Lattice Algorithms

BKZ. The Block Korkine-Zolotarey (BKZ) algorithm, which can be regarded as an extended version of the LLL algorithm [26], was first introduced by Schnorr and Euchner [33]. It uses an oracle that solves the SVP in the lattice with a block size of β , spanned by $\mathbf{B}_{[0, \beta-1]}$. The short vector is then recursively inserted into the lattice basis. A BKZ tour is initiated by calling an SVP-solver on consecutive blocks $\mathbf{B}_{[i, \min(i+\beta, d-1)]}$ for $i = 0, \dots, d-2$. The algorithm will proceed with these tours until there are no further changes observed within a single tour. We abbreviate BKZ with a block size of β as $\text{BKZ-}\beta$.

Lattice Sieving. Sieving algorithms are not only asymptotically superior to enumeration techniques [17, 19, 23, 33], but also showing better practical performance in higher lattice dimensions, due to the recent progress in both theory [8, 9, 21, 24] and practice [2, 15, 16, 18, 25].

The first sieving algorithm was proposed by Ajtai et al. [1] in 2001. It starts with a list of lattice vectors $L \subset \mathcal{L}$ and searches for shorter sums and differences of these vectors. The shorter combinations then replace the original longer vectors in the database. This process iterates until the database contains a significant number of short vectors, with the expectation of eventually finding the shortest vector.

Lattice sieving algorithms can be categorized into provably sieving algorithms and heuristic sieving algorithms. Heuristic sieving algorithms fall in the practical regime because of lower time and memory complexities. They are analyzed under the heuristic that the points in L are independently and uniformly distributed in a thin spherical shell. Nguyen and Vidick [30] proposed the first practical sieving algorithm, utilizing a database of $(4/3)^{d/2+o(d)} = 2^{0.2075d+o(d)}$ vectors and running in time $2^{0.415d+o(d)}$. The time complexity was subsequently improved to $2^{0.292d+o(d)}$ through nearest neighbor search techniques [8]. Various sieving algorithms have been efficiently implemented in G6K [2] and its GPU-accelerated version, G6K-GPU [16].

2.3 Hidden Number Problem

In the Hidden Number Problem (HNP) [12], we have an n -bit sized public modulus q , and there is a secret integer $\alpha \in \mathbb{Z}_q$, referred to as the hidden number. For $i = 0, 1, \dots, m-1$, t_i are uniformly random integers in \mathbb{Z}_q , and we are provided with the corresponding value a_i such that $|t_i \cdot \alpha - a_i|_q = k_i < q/2^l$. The problem is to recover the hidden number α when m samples (t_i, a_i) are given. We denote the above problem as $\text{HNP}(n, l)$.

2.4 Breaking ECDSA with Nonce Leakage

ECDSA. The global parameters for an ECDSA signature include an elliptic curve $E(\mathbb{F}_p)$ and a generator point G on $E(\mathbb{F}_p)$ of a prime order q . The secret signing key is an integer $0 \leq sk < q$, and the public verifying key is a point $[sk]G$. To sign a message hash h , the signer first generates a random integer nonce $0 \leq k < q$, then computes the signature $(r, s) = (([k]G)_x, |k^{-1} \cdot (h + sk \cdot r)|_q)$, where x subscript represents the x coordinate of the point.

ECDSA as a HNP. In a side-channel attack against ECDSA, the adversary may know l least significant bits of the nonce k . If we write $k = k_{msb} \cdot 2^l + k_{lsb}$, where $0 \leq k_{msb} < q/2^l$ and $0 \leq k_{lsb} < 2^l$, then we can obtain the following equation based on $s = |k^{-1} \cdot (h + r \cdot sk)|_q$:

$$2^{-l}(k_{lsb} - s^{-1} \cdot h) + k_{msb} = 2^{-l} \cdot s^{-1} \cdot r \cdot sk \pmod{q}.$$

This can be regarded as a HNP instance with $(t_i, a_i) = (|2^{-l} \cdot s^{-1} \cdot r|_q, |2^{-l} \cdot (k_{l_{sb}} - s^{-1} \cdot h)|_q)$ and hidden number $\alpha = sk$. In the case where the most significant bits of the nonce are leaked, the method of transformation into a HNP instance remains similar.

For convenience, we abbreviate ECDSA(n, l) as an n -bit ECDSA instance with l -bit leakage.

2.5 Solving the HNP with Lattices

To solve the HNP, in 1996, Boneh and Venkatesan [12] constructed the $(m + 1)$ -dimensional lattice generated by the rows of the following matrix:

$$\begin{bmatrix} q & 0 & \cdots & 0 & 0 \\ 0 & q & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & q & 0 \\ t_0 & t_1 & \cdots & t_{m-1} & 1/2^l \end{bmatrix}.$$

There exists a lattice vector $\mathbf{v} = (t_0 \cdot \alpha \bmod q, \dots, t_{m-1} \cdot \alpha \bmod q, \alpha/2^l)$. This lattice vector is close to the target vector $\mathbf{t} = (a_0, \dots, a_{m-1}, 0)$ since the distance $\|\mathbf{v} - \mathbf{t}\|$ can be bounded by $\sqrt{m+1} \cdot q/2^l$. If the distance is sufficiently small compared with other lattice vectors, the lattice vector can be found by solving the BDD problem using the nearest plane algorithm [7], or Kannan's embedding [23]. With Kannan's embedding, we construct the $(m + 2)$ -dimensional lattice basis

$$\begin{bmatrix} q & 0 & \cdots & 0 & 0 & 0 \\ 0 & q & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & q & 0 & 0 \\ t_0 & t_1 & \cdots & t_{m-1} & 1/2^l & 0 \\ a_0 & a_1 & \cdots & a_{m-1} & 0 & \tau \end{bmatrix},$$

where τ is the embedding number, which can be set to the upper bound of k_i , i.e., $\tau = q/2^l$. The target lattice vector becomes $\mathbf{v} = \pm(k_0, \dots, k_{m-1}, \alpha/2^l, -\tau)$ with a bounded norm of $\sqrt{m+2} \cdot q/2^l$.

Recentering Technique. In the definition of HNP, $0 \leq k_i < q/2^l$, for $i = 0, 1, \dots, m - 1$. Since the lattice can work for any sign of k_i , we can make a variable substitution $k'_i = k_i - w$ where $w = q/2^{l+1}$. The target vector becomes $(k'_0, k'_1, \dots, k'_{m-1}, \alpha/2^l, -\tau)$ and has a much shorter length. This technique can bring a significant improvement in practice and is widely used in the lattice attacks on HNP [3, 13, 28, 29, 38].

Elimination Method. Given a set of HNP equations $a_i + k_i = t_i \alpha \bmod q$, where $i = 0, 1, \dots, m - 1$, we can eliminate the variable α by substituting $\alpha = |(a_0 +$

$k_0)t_0^{-1}|_q$. This yields a new set of HNP equations. Incorporating the recentering technique, we have

$$a_i + w - (a_0 + w)t_0^{-1}t_i + k'_i = t_0^{-1}t_ik'_0 \pmod q,$$

where $i = 1, \dots, m-1$. This produces a transformed HNP instance $(t'_i, a'_i) = (t_0^{-1}t_i, a_i + w - (a_0 + w)t_0^{-1}t_i)$ with the transformed hidden number k'_0 . Then the new lattice is generated by:

$$\begin{bmatrix} q & 0 & \cdots & 0 & 0 & 0 \\ 0 & q & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & q & 0 & 0 \\ t'_1 & t'_2 & \cdots & t'_{m-1} & 1 & 0 \\ a'_1 & a'_2 & \cdots & a'_{m-1} & 0 & \tau \end{bmatrix},$$

referred to as the Albrecht and Heninger's lattice [3]. The lattice dimension, denoted as d , is reduced to $m+1$. The target lattice vector becomes $\mathbf{v} = \pm(k'_1, \dots, k'_{m-1}, k'_0, -\tau)$, with a bounded norm of $\sqrt{mw^2 + \tau^2}$. To find the target vector using a black box SVP solver, we expect the target vector to be the shortest vector, i.e., $\|\mathbf{v}\| \leq \sqrt{mw^2 + \tau^2} \leq \text{GH}(\mathcal{L})$. Combining this with $d = m+1$, we can estimate the minimal lattice dimension. Moreover, instead of using the upper bound of $\|\mathbf{v}\|$, Albrecht and Heninger considered the expected squared norm [3], i.e., $\mathbb{E}[\|\mathbf{v}\|^2] = mw^2/3 + m/6 + w^2$. Then the minimal lattice dimension can be estimated as the minimal integer d satisfying $\mathbb{E}[\|\mathbf{v}\|^2] \leq \text{GH}^2(\mathcal{L})$.

Sieving with Predicate. The sieving with predicate (Sieve-Pred) algorithm checks over the database generated by sieving algorithms using the predicate [3]. It does not treat the sieving algorithm as a black box SVP solver. This idea is inspired by Micciancio et al. [27], which suggests that the sieving algorithm not only outputs the shortest vector, but also provides all vectors with norm less than $\sqrt{4/3} \text{GH}(\mathcal{L})$, under specific heuristic assumptions. Algorithm 1 is expected to find the target vector under Assumption 1 stated below.

Assumption 1 ([15]) *When a 2-sieve algorithm terminates, it outputs a database L containing all vectors with norm $\leq \sqrt{4/3} \text{GH}(\mathcal{L})$.*

Theorem 1 ([3]). *Let $\mathcal{L} \subset \mathbb{R}^d$ be a lattice containing a vector v such that $\|\mathbf{v}\| \leq \sqrt{4/3} \text{GH}(\mathcal{L})$. Under Assumption 1, Algorithm 1 is expected to find the minimal \mathbf{v} satisfying $f(\mathbf{v}) = 1$ in $2^{0.292d+o(d)}$ steps and $(4/3)^{d/2+o(d)}$ calls to the predicate $f(\cdot)$.*

Algorithm 1: Sieving with Predicate

Input: Lattice $\mathcal{L}(\mathbf{B})$, predicate $f(\cdot)$ **Output:** \mathbf{v} such that $\|\mathbf{v}\| \leq \sqrt{4/3} \text{GH}(\mathcal{L})$ and $f(\mathbf{v}) = 1$ or \perp

```

1  $\mathbf{r} \leftarrow \perp$  ;
2 Run the sieving algorithm on  $\mathcal{L}(\mathbf{B})$  and output list  $L$ ;
3 for  $\mathbf{v} \in L$  do
4   | if  $f(\mathbf{v}) = 1$  and ( $r = \perp$  or  $\|\mathbf{v}\| < \|\mathbf{r}\|$ ) then
5   |   |  $\mathbf{r} \leftarrow \mathbf{v}$ ;
6 return  $\mathbf{r}$ ;

```

Thus, using the Sieve-Pred algorithm, the minimal lattice dimension can be estimated as the minimal integer d satisfying $\mathbb{E}[\|\mathbf{v}\|^2] \leq 4 \text{GH}^2(\mathcal{L})/3$.

3 Improved Algorithms

In this section, we propose several new algorithms for solving the HNP. We decompose the hidden number k'_0 as $x \cdot \alpha_0 + \alpha_1$, and introduce a new lattice construction that uses x to trade off the lattice dimension. Through theoretical analysis, we demonstrate that our lattice can offer a reduction of approximately $(\log x)/l$ compared to the lattice presented by Albrecht and Heninger [3]. While the target vector contains information about α_0 , the lack of information about α_1 makes it impossible to directly compute the hidden number α . This makes previous predicates ineffective [3, 38]. Due to the idea of Sieve-Pred algorithm [3], we need to search within the exponentially large database output by the sieving algorithms. Therefore, an efficient process that excludes incorrect candidates is essential. To address this issue, we propose a prescreening technique, an interval reduction algorithm, and a linear predicate. Moreover, we show that under Assumption 1, there exists a constant $c > 0$ such that the success probability of our algorithm is at least c .

3.1 New Lattice Construction Based on Decomposition Technique

The main idea behind our attack is using more HNP samples to obtain equations with small t'_i . Having many samples with small t'_i allows us to increase the lattice determinant, while keeping the norm of the target vector roughly unchanged. This enables us to reduce the lattice dimension, while still satisfying the enabling condition $\mathbb{E}[\|\mathbf{v}\|^2] \leq 4/3 \text{GH}^2(\mathcal{L})$ for the attack.

Lattice Construction. Following Albrecht and Heninger's lattice [3], we construct the lattice generated by

$$\begin{bmatrix} q & 0 & \cdots & 0 & 0 & 0 \\ 0 & q & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & q & 0 & 0 \\ x \cdot t'_1 & x \cdot t'_2 & \cdots & x \cdot t'_{m-1} & y & 0 \\ a'_1 & a'_2 & \cdots & a'_{m-1} & 0 & \tau \end{bmatrix},$$

where $x, y > 0$, and τ are undetermined coefficients. This modification increases Albrecht and Heninger's original lattice volume by a factor of y .

In [34], Sun et al. decomposed the hidden number α as $\alpha = 2^c \cdot \alpha_0 + \alpha_1$, where $0 \leq \alpha_1 < 2^c$. We propose a more generalized form of decomposition to the new hidden number: $k'_0 = x \cdot \alpha_0 + \alpha_1$, where $|\alpha_1| \leq x/2$. Here, x is an arbitrary positive integer and $|\alpha_1|$ is 1-bit smaller. Then, for $i = 1, \dots, m-1$, we have $x \cdot t'_i \cdot \alpha_0 - a'_i \equiv k'_i - \alpha_1 \cdot t'_i \pmod{q}$. The target vector in the lattice becomes

$$\mathbf{v} = \pm(k'_1 - \alpha_1 \cdot t'_1, \dots, k'_{m-1} - \alpha_1 \cdot t'_{m-1}, y \cdot \alpha_0, -\tau).$$

To keep the norm of the target vector approximately the same as in Albrecht and Heninger's lattice [3], we need to make the ratio $r = \mathbb{E}[(k'_i - \alpha_1 t'_i)^2] / \mathbb{E}[(k'_i)^2]$ close to 1. Now, we show how to achieve this. The following analysis is based on the assumption that α_1 is independent of k'_i and a'_i .

Recall that k'_i and t'_i are uniformly distributed in $[-w, w)$ and $[-q/2, q/2)$, respectively. By setting the upper bound of $|t'_i|$ to be B , t'_i becomes uniformly distributed in $[-B, B)$. Moreover, to make α_1 close to a uniform distribution over $[-x/2, x/2)$, the condition $x \ll w$ should be satisfied. Thus, we can set a theoretical upper bound $w/2^{10} = q/(2^{l+11})$ for x . Under these conditions, we have:

$$\begin{aligned} r &= \frac{\mathbb{E}[(k'_i - \alpha_1 t'_i)^2]}{\mathbb{E}[(k'_i)^2]} = \frac{\mathbb{E}[(k'_i)^2] - 2\mathbb{E}[\alpha_1] \mathbb{E}[k'_i t'_i] + \mathbb{E}[\alpha_1^2] \mathbb{E}[t'^2_i]}{\mathbb{E}[(k'_i)^2]} \\ &= 1 + \mathbb{E}[\alpha_1^2] \frac{\mathbb{E}[t'^2_i]}{\mathbb{E}[(k'_i)^2]}. \end{aligned}$$

Note that if an integer variable k is uniformly distributed over $[-w, w)$, then $\mathbb{E}[k^2] = w^2/3 + 1/6$. Thus, we have

$$\begin{aligned} r &= 1 + \left(\frac{x^2}{12} + \frac{1}{6}\right) \left(\frac{2B^2 + 1}{2w^2 + 1}\right) \\ &\leq 1 + \left(\frac{x^2}{12} + \frac{1}{6}\right) \left(\frac{B^2}{w^2} + \frac{1}{2w^2 + 1}\right) \leq 1 + 2^{-20} + \left(\frac{x^2}{12} + \frac{1}{6}\right) \frac{B^2}{w^2}. \end{aligned}$$

It can be seen that as B decreases, r approaches 1. However, the expected number of required samples increases by a factor of $q/(2B)$. To balance keeping r close to 1 and minimizing the number of required samples, we set $B = w/(2^3 x) = q/(2^{l+4} x)$. With this setting, we only need $2^{l+3} x$ times the original number of samples, and r is approximately $1 + (x^2 B^2 + 2B^2)/(12w^2) = 1 + 1/768 + 1/(384x^2)$, which is close to 1.

Next, we focus on the selection of parameters to optimize the performance of our attack. Given that $|k'_0|$ is bounded by w , we know that $|\alpha_0| \leq w/x$. Then we have

$$\mathbb{E}[\|\mathbf{v}\|^2] = (m-1) \frac{w^2}{3} + y^2 \frac{w^2}{3x^2} + \tau^2.$$

Our goal is to minimize the ratio $\mathbb{E}[\|\mathbf{v}\|^2] / \text{GH}^2(\mathcal{L})$, where

$$\text{GH}^2(\mathcal{L}) = \frac{(m+1)}{2\pi e} \cdot q^{\frac{2(m-1)}{m+1}} \cdot y^{\frac{2}{m+1}} \cdot \tau^{\frac{2}{m+1}}.$$

Given x , according to the AM-GM inequality, we have

$$\begin{aligned} \mathbb{E} [\|\mathbf{v}\|^2] &= \underbrace{\frac{w^2}{3} + \dots + \frac{w^2}{3}}_{m-1} + y^2 \frac{w^2}{3x^2} + \tau^2 \\ &\geq (m+1) \left(\left(\frac{w^2}{3} \right)^{m-1} \cdot y^2 \frac{w^2}{3x^2} \cdot \tau^2 \right)^{1/(m+1)} \\ &= (m+1) \left(\frac{w^2}{3} \right)^{m/(m+1)} \cdot x^{-\frac{2}{m+1}} \cdot y^{\frac{2}{m+1}} \cdot \tau^{\frac{2}{m+1}}. \end{aligned}$$

Thus, $\mathbb{E} [\|\mathbf{v}\|^2] / \text{GH}^2(\mathcal{L})$ attains its minimum value when $y = x$ and $\tau = w/\sqrt{3}$.

Reduction of Lattice Dimension. The new lattice construction can lead to a reduction in the lattice dimension, which is described in Theorem 2. This significantly improves the efficiency of lattice-based attacks, as the time complexity of sieving algorithms increases exponentially with an increase in the lattice dimension d .

Theorem 2. *For any positive integer x and the number of leaked bits l , the reduction of lattice dimension between our lattice and Albrecht and Heninger's lattice is given by*

$$\frac{2 \log x}{2l + 3 - \log(\pi e)} \approx \frac{\log x}{l}.$$

Proof. According to Assumption 1, the lattice dimension is the minimal integer d satisfying $\mathbb{E} [\|\mathbf{v}\|^2] \leq 4/3 \cdot \text{GH}^2(\mathcal{L})$. In Albrecht and Heninger's lattice [3], let $\tau = w/\sqrt{3}$, then we have

$$\begin{aligned} \mathbb{E} [\|\mathbf{v}\|^2] &= m \left(\frac{w^2}{3} + \frac{1}{6} \right) + \tau^2 \approx d \cdot \frac{w^2}{3}, \\ \text{GH}^2(\mathcal{L}) &= \frac{d}{2\pi e} \cdot q^{\frac{2(d-2)}{d}} \cdot \left(\frac{w^2}{3} \right)^{\frac{1}{d}}. \end{aligned}$$

Substituting $w = q/2^{l+1}$, and taking the logarithm, we have

$$d \geq \frac{2l + 2 + 2 \log q + \log 3}{2l + 3 - \log(\pi e)}.$$

In our lattice, $\mathbb{E}[\|\mathbf{v}\|^2]$ remains approximately the same, but $\text{GH}^2(\mathcal{L})$ increases by a factor of $x^{2/d}$. Thus, the new lattice dimension d' needs to satisfy

$$d' \geq \frac{2l + 2 + 2 \log q + \log 3 - 2 \log x}{2l + 3 - \log(\pi e)}.$$

The reduction of lattice dimension is given by

$$\frac{2 \log x}{2l + 3 - \log(\pi e)} \approx \frac{\log x}{l}.$$

□

Figure 1 illustrates the reduction of lattice dimension as x increases. Four lines are plotted in Fig. 1, representing HNP(160, 1), HNP(192, 1), HNP(224, 1), and HNP(384, 2), respectively. Solving these instances is believed to be impractical by previous lattice-based approaches [3, 5, 6, 34, 38]. However, our experimental results in Sect. 5 demonstrate the feasibility of solving HNP(160, 1) by using a large x . More computational resources and samples are required for solving more difficult instances such as HNP(192, 1).

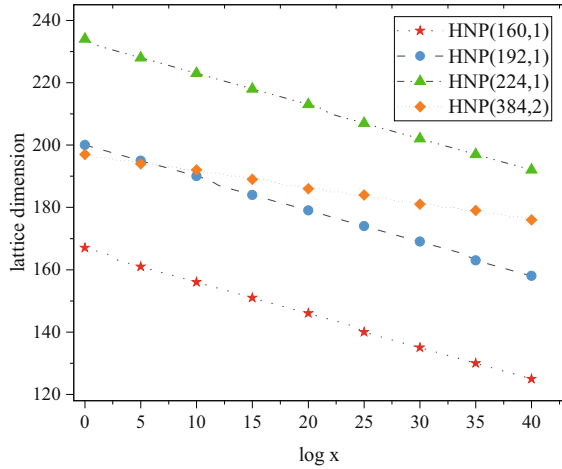


Fig. 1. Lattice dimension and x .

Success Probability. Under Assumption 1, the sieving algorithm outputs all vectors satisfying $\|\mathbf{v}\|^2 \leq 4 \text{GH}^2(\mathcal{L})/3$. Consequently, the probability that the sieving algorithm can output the target vector is represented as $\Pr(\|\mathbf{v}\|^2 \leq \mathbb{E}[\|\mathbf{v}\|^2])$, which is the success probability of our algorithm.

Theorem 3. *Let \mathbf{v} be the target vector of our lattice described in Sect. 3.1. For all $d \geq 3$, there exists a constant $c > 0$ such that $\Pr(\|\mathbf{v}\|^2 \leq \mathbb{E}[\|\mathbf{v}\|^2]) \geq c$.*

Proof. Recall the vector representation

$$\begin{aligned} \mathbf{v} &= (k'_1 - \alpha_1 t'_1, \dots, k'_{m-1} - \alpha_1 t'_{m-1}, x\alpha_0, -w/\sqrt{3}) \\ &= (v_0, v_{d-3}, v_{d-2}, v_{d-1}). \end{aligned}$$

For k'_i , t'_i , α_0 , and α_1 , they are all uniformly distributed: k'_i is over $[-w, w)$, t'_i is over $[-w/(8x), w/(8x))$, α_0 is over $[-w/x, w/x)$, and α_1 is over $[-x/2, x/2)$. It follows that v_0, \dots, v_{d-3} are independent and identically distributed.

Let $\mathbb{P}_d = \Pr\left(\frac{1}{d-2} \sum_{i=0}^{d-3} v_i^2 \leq \mathbb{E}[v_0^2]\right)$. Then we have

$$\begin{aligned} \Pr(\|\mathbf{v}\|^2 \leq \mathbb{E}[\|\mathbf{v}\|^2]) &= \Pr\left(\sum_{i=0}^{d-3} v_i^2 + x^2 \alpha_0^2 \leq (d-2)\mathbb{E}[v_0^2] + \frac{w^2}{3}\right) \\ &\geq \Pr\left(\sum_{i=0}^{d-3} v_i^2 \leq (d-2)\mathbb{E}[v_0^2]\right) \cdot \Pr\left(x^2 \alpha_0^2 \leq \frac{w^2}{3}\right) \\ &= \frac{\sqrt{3}}{3} \Pr\left(\frac{1}{d-2} \sum_{i=0}^{d-3} v_i^2 \leq \mathbb{E}[v_0^2]\right) = \frac{\sqrt{3}}{3} \mathbb{P}_d. \end{aligned}$$

Since $\text{Var}[v_0^2] < +\infty$, by the Central Limit Theorem, it holds that

$$\lim_{d \rightarrow +\infty} \Pr\left(\frac{1}{d-2} \sum_{i=0}^{d-3} v_i^2 \leq \mathbb{E}[v_0^2]\right) = \frac{1}{2}.$$

Hence, there exists a positive constant $c_1 > 0$ such that $\mathbb{P}_d \geq c_1$ for any $d \geq 3$. Let $c = c_1/\sqrt{3}$. Finally, we get

$$\Pr(\|\mathbf{v}\|^2 \leq \mathbb{E}[\|\mathbf{v}\|^2]) \geq \frac{\sqrt{3}}{3} \mathbb{P}_d \geq c.$$

□

3.2 Improved Linear Predicate

In Albrecht and Heninger's approach, they employ non-linear constraints as a predicate to determine the unique hidden number [3, 4]. Their predicate initially checks whether the absolute value of the last element of the candidate vector is τ . Subsequently, it determines the target vector by checking whether r is equal to $([k]G)_x$, where G is the generator point on the curve, r is a specific signature, and k is the corresponding nonce that can be computed from the candidate vector. The predicate involves time-consuming scalar multiplication on the curve.

In this section, we propose an improved linear predicate that utilizes linear constraints from $2 \log q$ HNP samples and only involves two vector inner products, providing a significant efficiency advantage. Moreover, we present a modified version of the Sieve-Pred algorithm to integrate our predicate, and achieve higher efficiency.

Linear Predicate. Our linear predicate is described in Algorithm 2. It operates on a 2-dimensional vector $\mathbf{v} = (v_0, v_1)$, representing the last two elements of a candidate vector in the database. This predicate determines whether the candidate vector satisfies a set of linear conditions. If these conditions are met, the predicate reveals the hidden number; otherwise, it returns \perp . The algorithm follows these steps:

- (1) Check if $0 < |v_0| \leq w$ and whether $|v_1|$ equals $\pm\tau$. If this condition is met, proceed to the next step; otherwise, return \perp .
- (2) Calculate the candidate α' from \mathbf{v} and the HNP sample (t_0, a_0) .
- (3) For N HNP samples (t_i, a_i) , check whether $|t_i \cdot \alpha' - a_i|_q < q/2^l$ for $i = 0, \dots, N - 1$. If this condition holds, return the candidate α' as the correct hidden number; otherwise, return \perp .

Note that the HNP samples used in step (3) are distinct from those used in our lattice construction.

Algorithm 2: Improved Linear Predicate

Input: A 2-dimensional vector $\mathbf{v} = (v_0, v_1)$, modulus q , number of nonce leakage l , embedding number τ , $N = 2 \log q$ HNP samples (t_i, a_i)

Output: The hidden number α or \perp

```

1 if  $v_0 = 0$  or  $|v_0| > q/2^{l+1}$  or  $|v_1| \neq \tau$  then
2   | return  $\perp$ ;
3  $k_0 \leftarrow -\text{sign}(v_1) \cdot v_0 + q/2^{l+1}$ ;
4  $\alpha' \leftarrow t_0^{-1} \cdot (a_0 + k_0) \bmod q$ ;
5 for  $i = 0$  to  $N - 1$  do
6   | if  $|t_i \cdot \alpha' - a_i|_q \geq q/2^l$  then
7     | | return  $\perp$ ;
8 return  $\alpha'$ ;
    
```

Theoretical Analysis of Linear Predicate. According to the definition of HNP, the correct candidate α' should satisfy all the constraints from these HNP samples. However, for an incorrect candidate, there still exists a small chance of meeting all the constraints. Consider the case where Algorithm 2 receives candidate values for the hidden number α' from the range $[0, q - 1]$, rather than computing alpha from the vectors obtained by the sieving algorithm. In this case, α' is uniformly distributed over $[0, q - 1]$. When t_i and a_i are fixed, $|t_i \alpha' - a_i|_q$ is also uniformly distributed over $[0, q - 1]$. Therefore, for $i = 0, 1, \dots, N - 1$, we have

$$\Pr(|t_i \cdot \alpha' - a_i|_q < q/2^l) = 2^{-l}.$$

Let $N = 2 \log q$. Then an incorrect candidate α' satisfies all the constraints with probability $2^{-2l \log q} = q^{-2l}$. Since there are $q - 1$ incorrect candidate α' , the probability that the algorithm can find the candidate in the interval $[0, q - 1]$ is given by

$$(1 - q^{-2l})^{q-1} \geq 1 - (q - 1)q^{-2l} \geq 1 - \frac{q - 1}{q^2} = 1 - \text{negl}(\log q).$$

Here, we prove that the probability of the algorithm correctly identifying all q inputs $\{0, \dots, q - 1\}$ is very close to 1. Consequently, it can also correctly identify whether the hidden number candidates obtained from the lattice are correct.

The expected number of verifying operations for each candidate can be calculated as follows. As mentioned earlier, the samples used in lattice construction and linear predicate are distinct from each other. This allows us to assume that the values of $|t_i\alpha - a_i|_q$ are uniformly distributed over the range $[0, q - 1]$. Consider a scenario with M candidates, under this assumption, the expected number of candidates meeting the first constraint is $M/2^l$, as only a fraction of $1/2^l$ candidates will satisfy the condition. Similarly, the expected number of candidates meeting the second constraint is $M/2^{2l}$, and so on. Therefore, the total expected number of verifying operations needed for M candidates is $M + M/2^l + M/2^{2l} + \dots = M/(1 - 2^{-l})$. Consequently, only $1/(1 - 2^{-l})$ verifying operations on average are needed for each candidate, and the parameter $2 \log q$ does not affect the efficiency of our predicate.

Comparison with the Predicate in [38]. In [38], Xu et al. introduce a linear predicate that requires a d -dimensional vector as input. In the sieving implementation G6K [2], vectors in the output database are represented as coordinates under the lattice basis. To obtain all positions of the candidate vector, one needs to perform d vector inner products, i.e., multiply the d -dimensional coordinate vector by the $d \times d$ lattice basis matrix. While our predicate only involves two vector inner products, which provides a notable efficiency advantage in practice.

In addition, our predicate uses linear constraints from new HNP samples, rather than those used in the lattice construction². The reason is that the vector \mathbf{v} in the sieving database is inherently shorter. If HNP samples (t_i, a_i) from the lattice construction are used, the probability $\Pr(|t_i \cdot \alpha' - a_i| < q/2^{l+1})$ (equivalent to $\Pr(|v_i| < q/2^{l+1})$) will be higher. Consequently, more linear constraints need to be verified to identify a false candidate.

Furthermore, we would like to point out an issue with the predicate in [38]. Their predicate may return true for some incorrect candidates, which is not as expected. The underlying reason can be explained as follows: in Albrecht and Heninger's lattice, for any $\theta \in \mathbb{N}^+$, consider the following lattice vector:

$$v_\theta = (|\theta t'_1 - a'_1|_q, \dots, |\theta t'_{d-2} - a'_{d-2}|_q, \theta, -\tau)$$

The predicate in [38] first checks whether v_{d-1} is equal to $\pm\tau$, and then computes the candidate α as $t_0^{-1}(\theta + q/2^{l+1} + a_0)$. Then it continues to check whether $a_i + v_{i-1} + q/2^{l+1}$ is equal to $t_i\alpha$ (Lines 6 and 13 in Algorithm 3 [38]). However, we always have $a_i + v_{i-1} + q/2^{l+1} = t_i\alpha \pmod q$. Specifically, we compute:

$$a_i + v_{i-1} + q/2^{l+1} = a_i + q/2^{l+1} + \theta t'_i - a'_i$$

Substituting $a'_i = a_i + q/2^{l+1} - (a_0 + q/2^{l+1})t_0^{-1}t_i$ and $t'_i = t_0^{-1}t_i \pmod q$ into the above equation, we get:

$$a_i + v_{i-1} + q/2^{l+1} = t_i t_0^{-1}(\theta + q/2^{l+1} + a_0) = t_i\alpha \pmod q$$

² In fact, the samples used in the lattice construction, the linear predicate, the interval reduction algorithm, and the prescreening technique are all distinct from each other.

This indicates that the predicate in [38] will return true for any θ and vector v_θ . To enhance performance, the conditions in Lines 6 and 13 of Algorithm 3 [38] could be modified to $|v_{i-1}|_q > q/2^{l+1}$.

Modified Sieving with Predicate. As we have made modifications to both the input and output of the predicate, a modified version of the Sieve-Pred algorithm is proposed to ensure compatibility with the new predicate. The algorithm is outlined in Algorithm 3. It no longer takes the entire lattice vector as input but only the last two elements. Moreover, the algorithm immediately outputs the solution when our linear predicate returns true, instead of searching the entire database.

Algorithm 3: Modified Sieving with Predicate

Input: Lattice $\mathcal{L}(\mathbf{B})$ with dimension d , predicate $f(\cdot)$

Output: The hidden number α or \perp

- 1 Run the sieving algorithm on $\mathcal{L}(\mathbf{B})$ and output list L ;
 - 2 **for** $v \in L$ **do**
 - 3 $\alpha \leftarrow f(v_{d-2}, v_{d-1})$;
 - 4 **if** $\alpha \neq \perp$ **then**
 - 5 **return** α ;
 - 6 **return** \perp ;
-

3.3 Predicate for Decomposition Technique

In Sect. 3.1, the transformed hidden number k'_0 is decomposed as $\alpha_0 x + \alpha_1$ where $|\alpha_1| \leq x/2$, and the target vector is $\mathbf{v} = \pm(k'_1 - \alpha_1 t'_1, \dots, k'_{m-1} - \alpha_1 t'_{m-1}, x\alpha_0, -\tau)$. This vector contains information about α_0 , which is a part of k'_0 . However, due to the absence of information about α_1 , it is impossible to directly compute the entire k'_0 . Consequently, previous predicates are ineffective in this scenario [3, 38].

The straightforward approach to recover k'_0 is to perform an exhaustive search over all possible values of α_1 , which has time complexity $O(x)$. For each candidate value of α_0 , we need to check the predicate for x candidate values of the hidden number. An exhaustive search will result in a substantial time overhead and becomes impractical when x is large. To address this issue, we introduce an interval reduction algorithm that reduces the complexity from $O(x)$ to $O(\log^2 x)$. Based on this algorithm, a predicate for the decomposition technique is also proposed.

There are two necessary notations. Assume that R is the union of a set of intervals. Let $|R|$ be the number of intervals in R , and $\|R\|$ be the number of integers within the intervals in R . For example, if $R = \{[1, 4]\}$, then $|R| = 1$ which means that there is one interval in R , and $\|R\| = 4$ which means that there are four integers within the interval $[1, 4]$.

Interval Reduction Algorithm. The interval reduction algorithm takes an interval of length M and $\log M$ transformed HNP samples as input, and produces

a set of smaller intervals as output. We denote the input interval as $[\text{low}, \text{high}]$, where $M = \text{high} - \text{low} + 1$. Let R be the set of intervals output by this algorithm. The interval reduction algorithm guarantee that if the hidden number is in the input interval $[\text{low}, \text{high}]$, then the hidden number must be in one interval of R . This algorithm is described in Algorithm 4, and it contains two steps as follows.

Algorithm 4: Interval Reduction Algorithm

Input: interval $[\text{low}, \text{high}]$ that may contain the hidden number k'_0 , modulus q , parameter l , $N = \log M$ transformed samples (t'_i, a'_i) satisfying $t'_i = O(q/M)$

Output: set R of intervals that may contain k'_0

```

1  $R \leftarrow \{[\text{low}, \text{high}]\};$ 
2 for  $i = 0$  to  $N - 1$  do
3   | Generate a new interval set  $R_{\text{new}}$  based on  $i$ -th sample  $(t'_i, a'_i)$ ;
4   |  $R \leftarrow \text{IntervalSetIntersection}(R, R_{\text{new}})$ ;
5 return  $R$ ;
```

Firstly, we generate the intervals based on $\log M$ transformed samples (t'_i, a'_i) . These intervals are computed using the HNP equations and the interval $[\text{low}, \text{high}]$. It holds that

$$t'_i \cdot \text{low} \leq t'_i k'_0 = a'_i + k'_i + nq \leq t'_i \cdot \text{high},$$

for $i = 1, \dots, \log M$. Since $-w \leq k'_i \leq w - 1$, we get a set \mathcal{S} that contains all possible values of n . For a specific n_1 in \mathcal{S} , we have

$$t'_i k'_0 = a'_i + k'_i + n_1 q \in [a'_i + n_1 q - w, a'_i + n_1 q + w - 1].$$

Therefore,

$$t'_i k'_0 \in \bigcup_{n \in \mathcal{S}} [a'_i + nq - w, a'_i + nq + w - 1].$$

This process yields a set of intervals that are sorted in ascending order, and one of these intervals may contain k'_0 . To limit the number of intervals in this set, we require that $t'_i = O(q/M)$. For the rationale of this requirement, the reader is referred to the proof of Theorem 4.

Secondly, we intersect all the sets of intervals generated from the $\log M$ samples. For this operation, we present an interval set intersection algorithm in Algorithm 5. It takes two sets of intervals that are in ascending order as input and outputs their intersection. The time complexity of Algorithm 5 is $O(m + n)$, where m and n are the numbers of intervals in two sets respectively.

Predicate for Decomposition Technique. Our predicate for the decomposition technique is described in Algorithm 6. Firstly, this algorithm verifies

whether $|v_0| < q/2^{l+1}$ and whether $|v_1|$ equals $\pm\tau$. Secondly, it computes the interval $[\text{low}, \text{high}]$ that may contain k'_0 . Thirdly, the interval reduction algorithm is employed to generate a set R . Finally, an exhaustive search is carried out to check the linear predicate for each integer in the intervals of R . The expected time complexity of this algorithm is $O(\log^2 x)$, as shown in Theorem 4.

Algorithm 5: Interval Set Intersection Algorithm

Input: Two sets of intervals in ascending order: A, B

Output: Intersection of A and B

```

1  $i, j \leftarrow 0$ ;
2  $R \leftarrow \{\}$ ;
3 while  $i < |A|$  and  $j < |B|$  do
4   | Let  $[a_0, a_1]$  and  $[b_0, b_1]$  be the  $i$ -th and  $j$ -th intervals in  $A$  and  $B$ ,
   | respectively;
5   | if  $a_1 < b_0$  then
6   |   |  $i \leftarrow i + 1$ ;
7   |   | Continue;
8   | if  $b_1 < a_0$  then
9   |   |  $j \leftarrow j + 1$ ;
10  |   | Continue;
11  | if  $a_1 \geq b_1$  then
12  |   |  $j \leftarrow j + 1$ ;
13  |   | Add the interval  $[\max(a_0, b_0), b_1]$  to  $R$ ;
14  |   | Continue;
15  |   |  $i \leftarrow i + 1$ ;
16  |   | Add the interval  $[\max(a_0, b_0), a_1]$  to  $R$ ;
17 return  $R$ ;
```

Theorem 4. *The expected time complexity of Algorithm 6 is $O(\log^2 x)$.*

Proof. There are two parts in Algorithm 6. The first part is the interval reduction algorithm, and the second part is an exhaustive search.

(1) Let us study the time complexity of the first part. We need to bound $|R_{\text{new}}|$, where R_{new} is given in Algorithm 4. For any transformed sample (t'_i, a'_i) , we have both $nq + a'_i + w - 1 \geq t'_i \cdot \text{low}$ and $nq + a'_i - w \leq t'_i \cdot \text{high}$. Hence, the number of possible values of n is not bigger than

$$\frac{t'_i \cdot (\text{high} - \text{low}) + 2w - 1}{q} + 1 = \frac{t'_i \cdot x - t'_i + 2w - 1}{q} + 1.$$

Since $t'_i = O(q/x)$, there exists a constant C such that $|R_{\text{new}}| \leq C$.

For $i = 0, 1, \dots, N - 1$, let R_i be the set returned by the interval set intersection algorithm in Algorithm 4 just after i -th sample. Then we have

Algorithm 6: Predicate for Decomposition Technique

Input: A 2-dimensional vector $\mathbf{v} = (v_0, v_1)$, modulus q , number of nonce leakage l , embedding number τ , predicate $f(\cdot)$

Output: hidden number α or \perp

```

1 if  $v_0 = 0$  or  $|v_0| > q/2^{l+1}$  or  $|v_1| \neq \tau$  then
2   | return  $\perp$ ;
3 low  $\leftarrow -\text{sign}(v_1) \cdot v_0 - x/2$ , high  $\leftarrow -\text{sign}(v_1) \cdot v_0 + x/2$ ;
4  $R \leftarrow \text{IntervalReduction}([\text{low}, \text{high}])$ ;
5 for  $[a, b] \in R$  do
6   | for  $h = a$  to  $b$  do
7     |   |  $\alpha \leftarrow f(h, -\tau)$ ;
8     |   | if  $\alpha \neq \perp$  then
9     |   |   | return  $\alpha$ ;
10 return  $\perp$  ;
```

$|R_i| \leq (i + 1)C$. Thus, for any $0 \leq i \leq N - 1$, the time complexity of i -th step in Algorithm 4 is at most $O((i + 2)C)$. It follows that the time complexity of the first part is $O(N^2) = O(\log^2 x)$.

(2) For the time complexity of the second part, let $R_{-1} = [\text{low}, \text{high}]$. Then $\|R_{-1}\| = x$. For an incorrect hidden number candidate, the probability that this candidate satisfies the constraint given by each HNP sample is only $1/2^l$. Thus, for $0 \leq i \leq N - 1$, we have

$$\frac{\mathbb{E}(\|R_i\|)}{\mathbb{E}(\|R_{i-1}\|)} = \frac{1}{2^l}.$$

Finally, we get

$$\mathbb{E}(\|R_{N-1}\|) = \mathbb{E}(\|R_{-1}\|) \cdot \left(\frac{1}{2^l}\right)^N \leq \frac{x}{2^N} = 1.$$

Therefore, the expected time complexity of the second step is $O(1)$.

By (1) and (2), the expected time complexity of Algorithm 6 is $O(\log^2 x)$.

□

Pre-screening Technique. Before running the interval reduction algorithm in Algorithm 6, a pre-screening technique can be used to eliminate the majority of incorrect candidates. This technique involves only a few linear operations and can significantly enhance the efficiency of Algorithm 6.

The pre-screening technique makes use of a small number of transformed HNP samples (t'_i, a'_i) , where $|t'_i| \leq q/(2^{l+3}x)$. For an incorrect candidate α'_0 , it will be rejected if the following condition is satisfied:

$$\left| \left| x \cdot t'_i \cdot \alpha'_0 - a'_i + \frac{q}{2} \right|_q - \frac{q}{2} \right| > w + \frac{q}{2^{l+4}}.$$

Let us explain the reason. For any correct candidate α_0 , we have $x \cdot t'_i \alpha_0 - a'_i \equiv k'_i - \alpha_1 t'_i \pmod{q}$. Therefore, we get

$$|k'_i - \alpha_1 t'_i| \leq |k'_i| + |\alpha_1| \cdot |t'_i| \leq w + \frac{x}{2} \cdot \frac{q}{2^{l+3}x} = w + \frac{q}{2^{l+4}}.$$

Note that this technique does not increase the sampling cost, as we can use the samples that satisfy $q/(2^{l+4}x) \leq |t'_i| \leq q/(2^{l+3}x)$ for pre-screening. These samples are already available during the pre-selection of the samples used to construct the lattice in Sect. 3.1.

To illustrate the efficiency improvements brought by the interval reduction algorithm and pre-screening technique, we conduct an experiment on HNP(256, 2) with $x = 2^{15}$. We record the time taken to search the database using different methods on a single thread. The experimental data demonstrates that, compared with the exhaustive search, the interval reduction algorithm provides a 2590-fold speedup. Furthermore, when combined with the pre-screening technique, we achieve a 3895-fold speedup.

4 Hidden Number Problem with Erroneous Input

In practical side-channel attacks, errors often appear in the data. This means that attackers may obtain incorrect nonces, resulting in erroneous HNP samples. Lattice-based attacks are believed to perform poorly when dealing with erroneous input [6, 31]. A common strategy to tackle this issue is to run the HNP solver on subsets of the samples until a correct solution is found [22]. However, this method does not fundamentally improve the lattice’s ability to handle errors. Consequently, some works assume that the input is error-free [22, 34, 38]. In [3], Albrecht and Heninger discussed the solution for handling errors, but did not provide a detailed analysis. With the increase of error rate, the dimension of their lattice would increase rapidly. The restricted efficiency of their non-linear predicate would result in a high cost for searching the sieving database, thereby constraining the ability to handle erroneous HNP instances. On the other hand, Fourier analysis-based attacks demonstrated stronger robustness to errors [5, 6, 14], highlighting a gap between these two approaches.

In this section, we define HNP with erroneous input based on the ECDSA nonce leakage model. The effectiveness of our new lattice construction in solving this problem is demonstrated through theoretical analysis. An estimation for the minimum lattice dimension is also provided. Furthermore, we extend our algorithms in Sect. 3, and significantly enhance the lattice’s ability to handle errors. This narrows the gap between lattice-based attacks and Fourier analysis-based attacks.

4.1 Theoretical Analysis

Definition 4 (Hidden Number Problem with Erroneous Input). *Given a modulus q , an error rate $0 < p < 1$, and both the hidden number α and the*

coefficients t_i being random numbers in \mathbb{Z}_q , the elements a_i satisfy the condition that with probability $1 - p$, $|t_i\alpha - a_i|_q < q/2^l$, while with probability p , $|t_i\alpha - a_i|_q$ is a random number in \mathbb{Z}_q . The problem is to recover the hidden number α when m samples (t_i, a_i) are given.

Let us show the rationality of this definition. In the ECDSA signature, let $k = 2^l k_{msb} + k_{lsb}$, where $0 \leq k_{msb} < q/2^l$, and $0 \leq k_{lsb} < 2^l$. Then we have

$$2^{-l}(k_{lsb} - s^{-1} \cdot h) + k_{msb} \equiv 2^{-l}s^{-1}r \cdot sk \pmod{q}.$$

Assume that we have probability $1 - p$ to obtain the correct value of k_{lsb} , and probability p to obtain a random integer k'_{lsb} in $[0, 2^l - 1]$.

Let $\alpha = sk$, $a_i = |2^{-l}(k_{lsb} - s^{-1} \cdot h(m))|_q$, $k_i = k_{msb}$, and $t_i = |2^{-l}s^{-1}r|_q$. If we obtain a random integer k'_{lsb} , then $a_i = |2^{-l}(k'_{lsb} - s^{-1} \cdot h)|_q$, and we have

$$|t_i\alpha - a_i|_q = |k_{msb} + 2^{-l}(k_{lsb} - k'_{lsb})|_q = |2^{-l}(k - k'_{lsb})|_q.$$

Since k is randomly chosen from \mathbb{Z}_q and q is a prime number, $|2^{-l}(k - k'_{lsb})|_q$ is also a random number in \mathbb{Z}_q . Thus, we obtain a HNP instance with erroneous input.

New Minimum Lattice Dimension. In this section, the lattice constructed by us is the same as that in Sect. 3.1:

$$\begin{bmatrix} q & 0 & \cdots & 0 & 0 & 0 \\ 0 & q & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & q & 0 & 0 \\ x \cdot t'_1 & x \cdot t'_2 & \cdots & x \cdot t'_{m-1} & x & 0 \\ a'_1 & a'_2 & \cdots & a'_{m-1} & 0 & \tau \end{bmatrix}.$$

The target vector is $\mathbf{v} = \pm(k'_1 - \alpha_1 t'_1, \dots, k'_{m-1} - \alpha_1 t'_{m-1}, x\alpha_0, \tau)$. From the definition above, for $i = 0, \dots, m-1$, k_i is randomly distributed in $[0, q)$ with probability p . Thus, $x\alpha_0 \equiv k'_0 - \alpha_1 \equiv k_0 - w - \alpha_1$ and $k'_i - \alpha_1 t'_i \equiv k_i - w - \alpha_1 t'_i$ also have probability p of being randomly distributed in $[-q/2, q/2)$. So we have

$$\begin{aligned} \mathbb{E} [\|\mathbf{v}\|^2] &= (d-1)(1-p)\frac{w^2}{3} + (d-1)\frac{pq^2}{12} + \tau^2 \\ &= (d-1)\frac{w^2}{3} (1 + p(2^{2l} - 1)) + \tau^2. \end{aligned}$$

Similar to the analysis in Sect. 3.1, when $\tau^2 = w^2(1 + p(2^{2l} - 1))/3$, the ratio $\mathbb{E} [\|\mathbf{v}\|^2] / \text{GH}^2(\mathcal{L})$ obtains its minimum. Substituting this into $\mathbb{E} [\|\mathbf{v}\|^2] \leq 4 \text{GH}^2(\mathcal{L})/3$, we get

$$d \geq \frac{2l + 2 + 2 \log q + \log 3 - 2 \log x - \log(1 + p \cdot (2^{2l} - 1))}{2l + 3 - \log(\pi e) - \log(1 + p \cdot (2^{2l} - 1))}.$$

The reduction of dimension is $2 \log x / (2l + 3 - \log(\pi e) - \log(1 + p \cdot (2^{2l} - 1)))$, which is more than $\log x / l$. This discovery reveals that our new lattice construction achieves a greater reduction (compared to Albrecht and Heninger’s lattice [3]) in lattice dimension in the presence of erroneous samples compared to error-free samples.

From a different perspective, the parameter x can be viewed as a balance to the error rate p . Given the lattice dimension d , p amplifies the target vector’s squared magnitude by $1 + p(2^{2l} - 1)$, while x amplifies $\text{GH}^2(\mathcal{L})$ by $x^{2/d}$. To maintain this ratio, we can set $1 + p(2^{2l} - 1) = x^{2/d}$, which leads to $x = (1 + p(2^{2l} - 1))^{d/2}$. Thus, for a higher error rate, we can increase x to keep the lattice dimension unchanged.

4.2 Modified Algorithms

Linear Predicate for Erroneous Input. For the hidden number candidate α' , we compute $|t_i \alpha' - a_i|_q$ for each HNP sample (t_i, a_i) . If this value falls within the range $[0, q/2^l)$, we say α' passes the test; otherwise, it fails. For error-free samples, if the candidate fails a single test, it can be regarded as incorrect. However, if there are errors in the samples, this judgment is not necessarily true. Table 2 lists the passing probabilities for error-free and erroneous samples. We can see that regardless of whether the candidate is correct or not, there exists a possibility that it fails a test. We denote the probability of passing a single sample as p_1 when $\alpha' = \alpha$, and as p_2 when $\alpha' \neq \alpha$. Then we get $p_1 = 1 - p + p \cdot 2^{-l}$, $p_2 = 2^{-l}$, and $p_1 > p_2$.

To extend the linear predicate in Sect. 3.2 to handle erroneous input, we count the number of samples that pass the test. Specifically, we collect $N = 2 \log q$ samples and count the number of samples that α' passes, denoted as M . If M exceeds $N(p_1 + p_2)/2$, we conclude that α' is the correct hidden number α . Otherwise, we discard it. This procedure is detailed in Algorithm 7.

Table 2. Passing probability for error-free and erroneous samples

	Error-free sample	Erroneous sample
$\alpha' = \alpha$	1	2^{-l}
$\alpha' \neq \alpha$	2^{-l}	2^{-l}

Theorem 5. *Algorithm 7 has an overwhelming success probability $1 - \text{negl}(\log q)$.*

Proof. Let P_1 be the probability that Algorithm 7 rejects a correct hidden number, and P_2 be the probability that Algorithm 7 accepts an incorrect candidate.

Algorithm 7: Predicate for erroneous input

Input: A 2-dimensional vector $\mathbf{v} = (v_0, v_1)$, modulus q , number of nonce leakage l , embedding number τ , $N = 2 \log q$ erroneous samples (t_i, a_i)

Output: The hidden number α or \perp

```

1 if  $v_0 = 0$  or  $v_0 > q/2^{l+1}$  or  $|v_1| \neq \tau$  then
2   | return 0;
3  $k_0 \leftarrow -\text{sign}(v_1)v_0 + q/2^{l+1}$ ;
4  $\alpha' \leftarrow t_0^{-1}(a_0 + k_0) \bmod q$ ;
5  $M \leftarrow 0$ ;
6 for  $i = 0$  to  $N - 1$  do
7   | if  $|t_i \alpha' - a_i|_q < q/2^l$  then
8     |   |  $M \leftarrow M + 1$ ;
9 if  $M > N(1 - p + (1 + p)2^{-l})/2$  then
10  | return  $\alpha'$ ;
11 else
12  | return  $\perp$ ;

```

We prove both P_1 and P_2 are negligible. Define

$$X_i = \begin{cases} 1, & \text{if } \alpha' \text{ passes the } i\text{-th sample;} \\ 0, & \text{if } \alpha' \text{ fails the } i\text{-th sample.} \end{cases}$$

When $\alpha' = \alpha$, X_i follows the Bernoulli distribution with probability p_1 , and when $\alpha' \neq \alpha$, X_i follows the Bernoulli distribution with probability p_2 . Let $S_N = \sum_{i=1}^N X_i$.

For the case of P_1 , let μ_1 be the expected value of S_N . Then $\mu_1 = p_1 N$. Let $\delta_1 = (p_1 - p_2)/(2p_1) \geq 1/(8p_1)$. By the Chernoff inequality, we have

$$P_1 = \Pr\left(S_N \leq \frac{N(p_1 + p_2)}{2}\right) < e^{-\mu_1 \frac{\delta_1^2}{2}} \leq e^{-\frac{p_1 N}{128 p_1^2}} \leq e^{-\frac{\log q}{64}}.$$

For the case of P_2 , let μ_2 be the expected value of S_N . Then $\mu_2 = p_2 N$. Let $\delta_2 = (p_1 - p_2)/(2p_2) = (1 - p)(2^l - 1)/2 \geq 2^{l-3}$. By the Chernoff inequality, we have

$$P_2 = \Pr\left(S_N > \frac{N(p_1 + p_2)}{2}\right) < e^{-\mu_2 \frac{\delta_2^2}{2}} \leq e^{-p_2 N \frac{2^{2l-6}}{2}} \leq e^{-2^{l-6} \log q} \leq e^{-\frac{\log q}{32}}.$$

□

Pre-screening Technique for Erroneous Input. The idea above can also be applied to the pre-screening technique. To achieve efficient pre-screening, we use $\log q$ samples that satisfy $|t'_i| < q/(2^{l+3}x)$. For each sample (t'_i, a'_i) , we compute $\left| |xt'_i \alpha_0 - a'_i + q/2|_q - q/2 \right|$. A sample (t'_i, a'_i) is called non-compliant if $\left| |xt'_i \alpha_0 - a'_i + q/2|_q - q/2 \right| > w + q/2^{l+4}$.

During the pre-screening, we cannot make any decision based only on a single non-compliant sample. Our strategy is to collect a set of samples, and make the decision when the number of non-compliant samples reaches a certain threshold. Different from Algorithm 7, the goal of pre-screening is to retain the correct hidden numbers, rather than to eliminate all incorrect candidates. Since the number of erroneous samples is at most $3p \log q$ with overwhelming probability, we discard the hidden number candidate if more than $3p \log q$ samples are non-compliant.

Sub-sampling Technique. In Sect. 3.3, we introduce an interval reduction algorithm for lattices constructed from the decomposition technique. This algorithm can efficiently perform an exhaustive search over an interval. However, it requires error-free samples. Otherwise, the algorithm may exclude the correct candidate. To overcome this limitation, we propose a sub-sampling technique. The specific steps are as follows.

- (1) Select $3 \log x/2$ samples to form a pool.
- (2) Draw $\log x$ samples from this pool, and apply Algorithm 6 to the candidate α'_0 , but replace the linear predicate in Algorithm 6 with the predicate for erroneous input in Algorithm 7. Return upon success.
- (3) Repeat the second step for γ times. If the hidden number is not found, the candidate is rejected.

Let P_1 be the probability that this algorithm rejects a correct hidden number candidate, and P_2 be the probability that this algorithm accepts an incorrect hidden number candidate. For any correct hidden number α , it has probability $p_1 = 1 - p + p \cdot 2^{-l}$ to pass a single sample. Hence, the second step has probability $p_1^{\log x}$ to return α . Therefore, $P_1 = (1 - p_1^{\log x})^\gamma$. For any incorrect hidden number α , if α is returned, α must be returned by Algorithm 7. By Theorem 5, we have $P_2 = \text{negl}(\log q)$.

In practice, for given p, x and l , we can adjust γ to minimize P_1 . For example, for the case of $p = 0.02, x = 2^{25}, l = 1$, let $\gamma = 10$, then we get $P_1 < 3 \times 10^{-7}$.

5 Key Recovery of ECDSA with Nonce Leakage

In this section, experimental results are provided to demonstrate the performance of our algorithms. Assume that the least significant bits of the nonce used for each signature are leaked. Our goal is to recover the secret signing key by solving the corresponding HNP instance.

Our implementation integrates the progressive sieving technique [15, 25], which starts at a low sieving dimension and increases the sieving dimension by 1 with each iteration. The process of searching the database in Algorithm 3 is parallelized for improving the efficiency. Moreover, before applying Algorithm 3 to the lattice basis constructed from HNP samples, we preprocess the basis with BKZ-20, which can randomize the lattice basis and increase the success

rate. In [3], Albrecht et al. used BKZ-(d-20) in their preprocessing step. However, the experimental results show that our preprocessing step is much more efficient and achieves a success rate close to [3].

To introduce the strategy of our attack on instances of varying difficulty, we categorize ECDSA instances into three classes. Specifically, this classification relies on the minimum lattice dimension d estimated via Albrecht and Heninger’s lattice. These three classes are denoted as Easy ($d \leq 100$), Medium ($100 < d \leq 140$), and Hard ($d > 140$). In practical applications, we may adjust the parameter x . This allows us to obtain an optimal balance between time consumption and the number of available samples.

5.1 Compared with Other Lattice-Based Attacks

Solving Easy Instances. Table 3a lists our attack results for several easy instances. These experiments are conducted using a single thread on an Intel Xeon Gold 6154 CPU with the G6K library [2]. For each instance, the average CPU-seconds and the success rate (s/r) are calculated from 100 experiments. Figure 2 illustrates the efficiency comparison between our algorithm and recent works [3, 34, 38]. It is evident that our algorithm exhibits a significant efficiency advantage. For example, when dealing with ECDSA(384, 4), Albrecht and Heninger [3] successfully conducted an attack in 49200s, and Xu et al. [38] improved the time to 11153s, whereas we only require 5583s. When targeting ECDSA(192, 2), our attack also demonstrates a 31-fold speedup compared to the attack performed by Albrecht and Heninger [3].

When $x = 1$, the key difference between our work and other works [3, 38] lies in the predicate used in the attack. We substitute our predicate with those in [3] and [38] separately, and record the time of searching the database on the same machine. We denote the time taken by our predicate as t_1 , and the time taken by the predicates in [3] and [38] as t_2 and t_3 , respectively. The speed-up ratio brought by our predicate is shown in Fig. 3.

The success rate of our attack can be further improved by increasing the lattice dimension. Taking ECDSA(160, 2) as an example, Fig. 4 illustrates how the success rate notably increases as the lattice dimension grows. We conduct experiments on 500 randomly generated instances. When the lattice dimension reaches 92, the success rate approaches 100%. This observation can be attributed to the fact that the majority of the $\|\mathbf{v}\|/\text{GH}$ ratios are below $\sqrt{4/3}$, which enables us to find the target vectors through sieving algorithms with high success rates. The experimental results also indicate that our algorithm can still handle instances with $\|\mathbf{v}\|/\text{GH}$ ratios greater than $\sqrt{4/3}$ with a lower success rate, rather than failing 100%.

Solving Medium Instances. Our attack results for medium instances are presented in Table 3b. These experiments are conducted on an Intel Xeon Platinum 8480+ CPU and two GeForce RTX 4090 GPUs. We employ G6K-GPU [16] to achieve high-performance sieving algorithms. Our attacks demonstrate high efficiency. Taking ECDSA(256, 2) as an example, when x is set to 1, the attack is

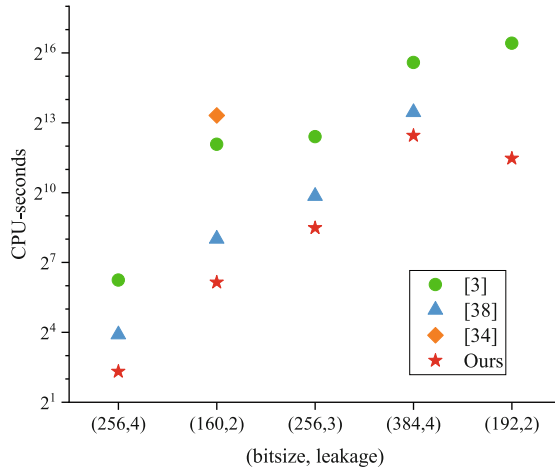
Table 3. Performance of our lattice-based attacks

Curve	Leakage	d	x	CPU-seconds	s/r	Previous records
secp160r1	2	82	1	206s	52%	259s in [38]
		77	2^{10}	71s	58%	
secp192r1	2	99	1	10360s	60%	87500s in [3]
		94	2^{10}	2829s	69%	
secp256r1	4	66	1	7s	65%	15s in [38]
		64	2^{10}	5s	79%	
	3	87	1	634s	53%	924s in [38]
		84	2^{10}	359s	57%	
secp384r1	4	98	1	8154s	62%	11153s in [38]
		96	2^{10}	5583s	56%	

(a) Easy instances

Curve	Leakage	d	x	Wall time	Mem GiB	Previous records
secp112r1	1	116	1	6min	35	260min in [38]
secp256r1	2	129	1	95min	219	466min in [38]
		124	2^{10}	31min	114	
secp384r1	3	130	1	128min	252	156min in [38]
		125	2^{15}	39min	132	

(b) Medium instances

**Fig. 2.** Comparison of CPU-seconds with previous works.

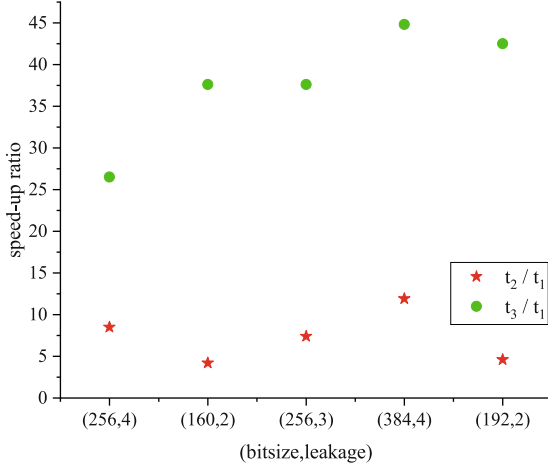


Fig. 3. Searching the sieving database using different predicates.

Table 4. New records of lattice-based attacks against ECDSA

Curve	Leakage	d	x	Samples	Wall time	Mem GiB
brainpoolp512r1	4	130	1	2^{10}	96min	254
(a) 4-bit leakage						
Curve	Leakage	d	x	Samples	Wall time	Mem GiB
secp128r1	1	131	1	2^8	72min	294
		118	2^{15}	2^{26}	8min	53
secp160r1	1	144	2^{14}	2^{25}	824min	1939
		138	2^{25}	2^{36}	279min	850
(b) 1-bit leakage						
Curve	Error rate	d	x	Samples	Wall time	Mem GiB
secp128r1	0.1	140	2^{20}	2^{31}	370min	1090
secp160r1	0.02	144	2^{14}	2^{25}	1009 min	1960
(c) Less than 1-bit leakage						

completed in 95 min by constructing a 129-dimensional lattice. The time can be significantly reduced by using a larger x . In our experiment, we set $x = 2^{10}$ and complete the attack in just 31 min. On the other hand, the attack in [38] required 466 min with four GeForce RTX 3090 GPUs.

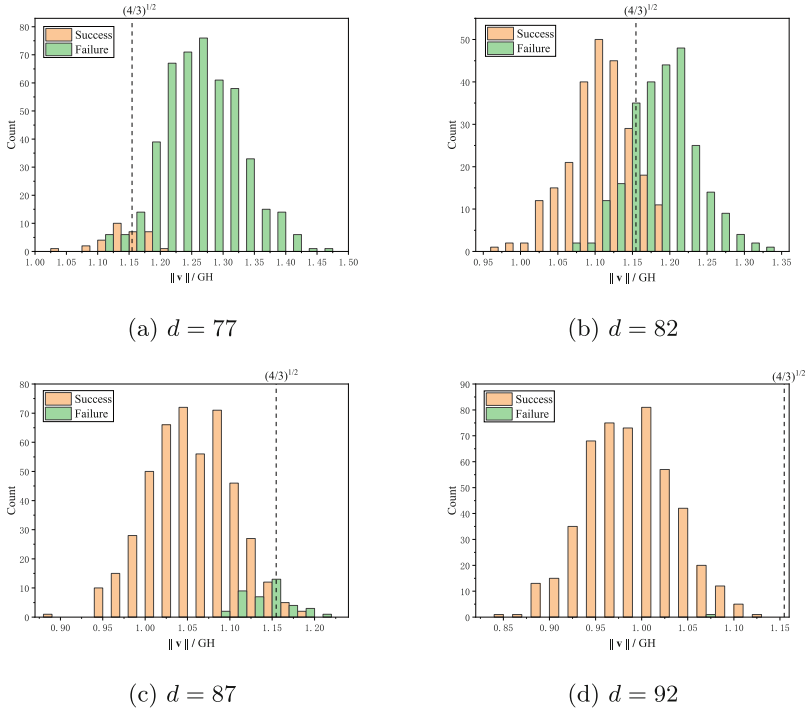


Fig. 4. Breaking ECDSA(160, 2) via lattices with different dimensions.

5.2 New Records of Lattice-Based Attacks Against ECDSA

We achieve several new records of lattice-based attacks against ECDSA. These attacks are conducted using an Intel Xeon Platinum 8480+ CPU and four GeForce RTX 4090 GPUs. Specific details about time and memory consumption can be found in Table 4.

4-Bit Leakage. The previous record for 4-bit leakage is only achieved on a 384-bit curve. As depicted in Table 4a, our algorithm is able to break ECDSA(512, 4) by a 130-dimensional lattice, which takes 96 min and consumes 254 GiB of memory.

1-Bit Leakage. The first implementation of Fourier-based attack against ECDSA with 1-bit leakage was proposed by Aranha et al. at ASIACRYPT 2014 [5]. They achieved a full key recovery on a 160-bit elliptic curve using 2^{33} ECDSA signatures. However, breaking ECDSA with 1-bit leakage using lattice algorithms has been considered to be very difficult. In 2023, Xu et al. reported a successful key recovery for 1-bit leakage on a 112-bit curve [38]. Nonetheless, breaking ECDSA(160, 1) was regarded as exceptionally challenging by previous lattice approaches [3, 5, 6, 34, 38]. Xu et al. required a lattice dimension of approximately 165, which would make the time and space complexities of sieving algorithms

unacceptable [38]. Besides, Sun et al. estimated the time complexity of their guessing bits algorithm is 2^{110} BKZ-30 operations [34].

We present the first implementation of lattice attacks against ECDSA with 1-bit leakage on both 160-bit and 128-bit curves. The experimental results are presented in Table 4b. When targeting ECDSA(160,1), it is essential to reduce the sieving dimension to a manageable size for accepted time and memory use. We implement the attack with the parameter x set to 2^{25} and 2^{14} , corresponding to total sample sizes of 2^{36} and 2^{25} , respectively. When x is set to 2^{25} , we construct a 138-dimensional lattice to perform the attack, which takes approximately 279 min and consumes 850 GiB of memory.

Less than 1-Bit Leakage. Before this work, only Fourier analysis-based attacks [6] could break ECDSA with less than 1-bit nonce leakage. We provide the first lattice-based attack results in Table 4c. Our attacks successfully handle an ECDSA instance with an error rate of 0.1 on a 128-bit curve and an ECDSA instance with an error rate of 0.02 on a 160-bit curve.

Acknowledgements. We would like to thank the anonymous reviewers of ASIACRYPT 2024, EUROCRYPT 2024 and CRYPTO 2024 for their insightful suggestions. We also thank Fan Huang, Xiaolin Duan, Yaqi Wang, and Changhong Xu for their valuable support to this work. This work was supported by National Natural Science Foundation of China (Grant No. 62472397) and Innovation Program for Quantum Science and Technology (Grant No. 2021ZD0302902).

References

1. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: 33rd ACM STOC. pp. 601–610. (2001). <https://doi.org/10.1145/380752.380857>
2. Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 717–746. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_25
3. Albrecht, M.R., Heninger, N.: On bounded distance decoding with predicate: Breaking the “lattice barrier” for the hidden number problem. In: Canteaut, A., Standaert, FX. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 528–558. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_19
4. Albrecht, M.R., Heninger, N.: Bounded distance decoding with predicate source-code (2020). <https://github.com/malb/bdd-predicate>
5. Aranha, D.F., Fouque, PA., Gérard, B., Kammerer, JG., Tibouchi, M., Zapalowicz, JC.: GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 262–281. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_14
6. Aranha, D.F., Novaes, F.R., Takahashi, A., Tibouchi, M., Yarom, Y.: LadderLeak: Breaking ECDSA with less than one bit of nonce leakage. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020, pp. 225–242. ACM Press (2020). <https://doi.org/10.1145/3372297.3417268>

7. Babai, L.: On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica* 6, 1-13 (1986). <https://doi.org/10.1007/BF02579403>
8. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th SODA, pp. 10-24. ACM-SIAM (2016). <https://doi.org/10.1137/1.9781611974331.ch2>
9. Becker, A., Gama, N., Joux, A.: Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. *Cryptology ePrint Archive, Report 2015/522* (2015). <http://eprint.iacr.org/2015/522>
10. Bleichenbacher, D.: On the generation of one-time keys in DL signature schemes. Presentation at IEEE P1363 Working Group Meeting (2000)
11. Bleichenbacher, D.: Experiments with DSA. Rump session at CRYPTO (2005). <https://www.iacr.org/conferences/crypto2005/r/3.pdf>
12. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In: Koblitz, N. (ed.) CRYPTO 96. LNCS, vol. 1109, pp. 129-142. Springer, Heidelberg (1996). <https://doi.org/10.1007/3-540-68697-5-11>
13. Breitner, J., Heninger, N.: Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 3-20. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-32101-7_1
14. De Mulder, E., Hutter, M., Marson, M.E., Pearson, P.: Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA. In: Bertoni, G., Coron, J.S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 435-452. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_25
15. Ducas, L.: Shortest vector from lattice sieving: A few dimensions for free. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 125-145. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-78381-9_5
16. Ducas, L., Stevens, M., van Woerden, W.: Advanced lattice sieving on GPUs, with tensor cores. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 249-279. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_9
17. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation* 44(170), 463-471 (1985)
18. Fitzpatrick, R., Bischof, C., Buchmann, J., Dagdelen, Ö., Göpfert, F., Mariano, A., Yang, B.-Y.: Tuning GaussSieve for speed. In: LATINCRYPT 2014. LNCS, vol. 8895, pp. 288-305. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-16295-9_16
19. Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 257-278. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_13
20. Heninger, N.: Using Lattices for Cryptanalysis. (2020). <https://simons.berkeley.edu/-talks/using-lattices-cryptanalysis>
21. Herold, G., Kirshanova, E., Laarhoven, T.: Speed-ups and time-memory trade-offs for tuple lattice sieving. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol 10769, pp.407-436. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76578-5_14
22. Jancar, J., Sedlacek, V., Svenda, P., Sys, M.: Minerva: The curse of ECDSA nonces. *IACR TCHES* 2020(4), 281-308 (2020). <https://doi.org/10.13154/tches.v2020.i4.281-308>

23. Kannan, R.: Minkowski's convex body theorem and integer programming. *Math. Oper. Res.* 12(3), 415-440 (1987). <https://doi.org/10.1287/moor.12.3.415>
24. Laarhoven, T.: Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In: Gennaro, R., Robshaw, M. (eds.) *CRYPTO 2015*. LNCS, vol. 9215, pp. 3-22. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_1
25. Laarhoven, T., Mariano, A.: Progressive lattice sieving. In: Lange, T., Steinwandt, R. (eds.) *PQCrypto 2018*. LNCS, vol. 10786, pp. 292-311. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-79063-3_14
26. Lenstra, A.K., Lenstra Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. *Mathematische Annalen* 261, 366-389 (1982). <https://infoscience.epfl.ch/record/164484/files/nscan4.PDF>
27. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: Charika, M. (ed.) *21st SODA*. pp. 1468-1480. ACM-SIAM (2010). <https://doi.org/10.1137/1.9781611973075.119>
28. Moghimi, D., Sunar, B., Eisenbarth, T., Heninger, N.: TPM-FAIL: TPM meets timing and lattice attacks. In: Capkun, S., Roesner, F. (eds.): *USENIX Security 2020*. pp. 2057-2073. (2020). <https://www.usenix.org/system/files/sec20-moghimi-tpm.pdf>
29. Nguyen, P.Q., Shparlinski, I.: The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology* 15(3), 151-176 (2002). <https://doi.org/10.1007/s00145-002-0021-3>
30. Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology* 2(2), 181-207 (2008). <https://doi.org/10.1515/JMC.2008.009>
31. Ryan, K.: Return of the hidden number problem. *IACR TCHES* 2019(1), 146-168 (2018). <https://tches.iacr.org/index.php/TCHES/article/view/7337>
32. Schnorr, C.P. : Lattice reduction by random sampling and birthday methods. In: Alt, H., Habib, M. (eds.) *STACS 2003*. LNCS, vol. 2607, pp. 145-156. Springer, Berlin, Heidelberg (2003). https://doi.org/10.1007/3-540-36494-3_14
33. Schnorr, C., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.* 66, 181-199 (1994). <https://doi.org/10.1007/BF01581144>
34. Sun, C., Espitau, T., Tibouchi, M., Abe, M.: Guessing bits: Improved lattice attacks on (EC)DSA with nonce leakage. *IACR TCHES* 2022(1), 391-413 (2022). <https://tches.iacr.org/index.php/TCHES/article/view/9302>
35. Takahashi, A., Tibouchi, M., Abe, M.: New Bleichenbacher records: Fault attacks on qDSA signatures. *IACR TCHES* 2018(3), 331-371 (2018). <https://tches.iacr.org/index.php/TCHES/article/view/7278>
36. The G6K development team: G6K (2020). <https://github.com/fplll/g6k>
37. The G6k-GPU-Tensor development team: G6k-GPU-Tensor (2021). <https://github.com/WvanWoerden/G6K-GPU-Tensor>
38. Xu, L., Dai, Z., Wu, B., Lin, D.: Improved attacks on (EC)DSA with nonce leakage by lattice sieving with predicate. *IACR TCHES* 2023(2), 568-586 (2023). <https://doi.org/10.46586/tches.v2023.i2.568-586>



Don't Use it Twice! Solving Relaxed Linear Equivalence Problems

Alessandro Budroni¹(✉), Jesús-Javier Chi-Domínguez¹, Giuseppe D'Alconzo², Antonio J. Di Scala², and Mukul Kulkarni¹

¹ Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, UAE
{alessandro.budroni,jesus.dominguez,mukul.kulkarni}@tii.ae

² Department of Mathematical Sciences, Polytechnic University of Turin, Turin, Italy
{giuseppe.dalconzo,antonio.discala}@polito.it

Abstract. The Linear Code Equivalence (LCE) Problem has received increased attention in recent years due to its applicability in constructing efficient digital signatures. Notably, the LESS signature scheme based on LCE is under consideration for the NIST post-quantum standardization process, along with the MEDS signature scheme that relies on an extension of LCE to the rank metric, namely the Matrix Code Equivalence (MCE) Problem. Building upon these developments, a family of signatures with additional properties, including linkable ring, group, and threshold signatures, has been proposed. These novel constructions introduce relaxed versions of LCE (and MCE), wherein multiple samples share the same secret equivalence. Despite their significance, these variations have often lacked a thorough security analysis, being assumed to be as challenging as their original counterparts. Addressing this gap, our work delves into the sample complexity of LCE and MCE—precisely, the sufficient number of samples required for efficient recovery of the shared secret equivalence. Our findings reveal, for instance, that one should not use the same secret twice in the LCE setting since this enables a polynomial time (and memory) algorithm to retrieve the secret. Consequently, our results unveil the insecurity of two advanced signatures based on variants of the LCE Problem.

Keywords: Algebraic Attack · Code Equivalence · Group Actions · Cryptanalysis · Post-quantum Cryptography

1 Introduction

Following the ongoing NIST post-quantum standardization process for additional digital signature schemes [28], there has been an increased interest in constructing new quantum-resistant digital signatures. Moving beyond the proposals at the prior NIST post-quantum standardization process [27], the research community explored a broader spectrum of computational problems, conjectured to be hard, for building efficient signature schemes. A family of such hard problems

is represented by those computational problems consisting of finding an equivalence or isomorphism between two algebraic/geometrical structures. For example, among the candidates for the NIST post-quantum standardization process, the digital signature Hawk [12] relies on the hardness of the Lattice Isomorphism Problem (LIP), LESS [2] on the Linear Code Equivalence Problem (LCE), MEDS [15] on the Matrix Code Equivalence Problem (MCE), and SQIsign [14] on the problem of finding isogenies between supersingular elliptic curves.

All these hard problems can be modeled as group actions. This common framework has been utilized by cryptographers in two significant ways. First, protocols defined for a specific hard problem have often been adapted into analogous protocols using another hard problem, leveraging the similar structure and properties of the underlying group actions. For example, the Calamari ring signature [10] relying on isogenies has been translated to code equivalence [4]. Second, some protocols have been defined in a general manner for group actions and subsequently instantiated with specific problems [7, 22, 25]. This approach not only broadens the applicability of these cryptographic protocols but also provides a unified theoretical foundation for their security and efficiency.

While group actions used in cryptography are generally assumed to guarantee one-wayness, specific group actions might or might not satisfy certain additional properties such as weak-unpredictability and weak-pseudorandomness. This subject has already been studied for LCE and MCE by D’Alconzo and Di Scala [19] and for LIP by Benčina et al. [8]. Consequently, instantiating protocols with a specific group action without ensuring that stronger cryptographic properties are satisfied may result in insecure protocols. In addition, to achieve specific functionalities such as threshold signature or linkability on ring signatures, some relaxed versions of these hard problems have been proposed. These variants are often conjectured to exhibit a level of difficulty comparable to their original counterparts but without formal proof or comprehensive cryptanalytic investigation.

In this work, we significantly improve the sample complexity estimated by D’Alconzo and Di Scala for LCE and MCE, i.e., the sufficient number of samples sharing the same secret required for breaking weak-unpredictability. Furthermore, we give an algorithm to solve two variants of LCE, namely the Inverse Code Equivalence Problem (ILCE) and the Code Equivalence Problem with two samples (2-LCE),¹ in polynomial time, that were introduced to construct linkable ring signatures [4] and threshold signatures [7], respectively. As a consequence, the schemes that rely on the hardness of ILCE and 2-LCE are not secure. However, we wish to highlight that our result does not affect the one-wayness of LCE/MCE.

We summarize in Table 1 the applications of LCE and MCE group actions that are still considered secure, and the ones that have been discovered not secure by this work and [19] since they require stronger properties such as weak-unpredictability and weak-pseudorandomness. For the case of constructing

¹ The authors in [7] gave a more general problem definition in terms of group actions, namely 2-Group Action Inverse Problem (2-GAIP). Here, we refer by 2-LCE to the 2-GAIP from [7] instantiated with LCE.

linkable ring signatures using the inverse problem of MCE, our work reveals that, algebraically, this problem is significantly weaker than classic MCE. Thus, we believe that further investigation in this case is necessary.

Table 1. Overview of the secure and insecure known instantiations of primitives constructed from LCE and MCE group actions. The symbols \times and \checkmark denote that the corresponding primitive is insecure or remains secure. The symbol $\checkmark(?)$ denotes that no specific attacks are known, but we suggest further investigation. The third column in the LCE setting concerns the cryptographic scenario when the code length doubles the code dimension.

	ID scheme/ signature	Commitment	Linkable ring signature from [4, 10, 16]	Pseudo random function from [1]	Updatable encryption from [25]
LCE	\checkmark	\checkmark	\times	\times	\times
MCE	\checkmark	\checkmark	$\checkmark(?)$	\times	\times

1.1 Overview of the Contribution

Informally, we say that two linear codes \mathcal{C}_1 and \mathcal{C}_2 of length n and dimension k over a finite field \mathbb{F}_q are equivalent if there exists a monomial matrix $\mathbf{Q} \in \mathbb{F}_q^{n \times n}$ such that $\mathcal{C}_2 = \mathcal{C}_1 \mathbf{Q}$. Given two generators $\mathbf{G}_1, \mathbf{G}_2 \in \mathbb{F}_q^{k \times n}$ of two equivalent codes, the Linear Code Equivalence Problem (LCE) is the problem of finding an invertible matrix \mathbf{S} and a monomial matrix \mathbf{Q} such that $\mathbf{G}_2 = \mathbf{S} \mathbf{G}_1 \mathbf{Q}$. When \mathbf{Q} is a permutation matrix, the problem is called Permutation Code Equivalence Problem (PCE).

On the Hardness of LCE Given t Samples

Our first contribution is investigating the impact of providing more than one LCE sample sharing the same secret matrix \mathbf{Q} to the adversary. We explore how this explicitly affects the hardness of recovering \mathbf{Q} . In particular, we derive a concrete bound on the number of necessary samples that allow an efficient recovery of the secret. Additionally, similar results are also obtained for MCE. We present our result in the following lemma:

Lemma (Informal). *For (n, k) -linear codes over a finite field \mathbb{F}_q , the secret monomial matrix \mathbf{Q} can be recovered from $\left\lfloor \frac{n^2}{k(n-k)} \right\rfloor + 1$ samples of LCE sharing the same \mathbf{Q} in polynomial time, with non-negligible probability.*

The above result improves upon the work by D’Alconzo and Di Scala [19], who provided a bound of $n \cdot k$ samples applicable solely to code generators that are not in systematic form. In contrast, our result removes this limitation, extending the applicability to codes represented in systematic form as well. The key ingredient of our result relies on constructing a linear system from each sample, where only the entries of \mathbf{Q} are the variables (and not those of \mathbf{S}). Specifically, we use the relation $\mathbf{G}_1 \mathbf{Q} \mathbf{H}_2^\top = \mathbf{0}$, where \mathbf{H}_2 is a parity-check matrix of \mathbf{G}_2 , to construct the following homogeneous linear system:

$$(\mathbf{G}_1 \otimes \mathbf{H}_2) \cdot \text{vec}(\mathbf{Q}) = \mathbf{0}, \quad (1)$$

where $\text{vec}(\mathbf{Q})$ is the column vector whose entries are the entries of \mathbf{Q} row-by-row. This linear system is underdetermined, meaning that there are fewer equations than variables. However, by combining the systems from different samples, we obtain with high probability a determined linear system whose solution can be found via Gaussian elimination, leading to the recovery of \mathbf{Q} .

Solving 2-LCE and ILCE for $(2k, k)$ -Linear Codes

Our second contribution is to introduce a polynomial-time algorithm for solving 2-LCE, i.e., the problem of retrieving \mathbf{Q} from only 2 LCE samples, specifically for $k = n/2$. Thanks to the fact that ILCE can be seen as a 2-LCE instance via a simple transformation, we are able to solve this problem in polynomial time as well. Our algorithm is inspired by Saeed’s work [33] and, in addition to the results mentioned in the lemma above, it exploits the structure of the secret monomial matrix to recover it.

Our method consists of first constructing a linear system as in Eq. (1) with the two available LCE samples

$$\begin{bmatrix} \mathbf{G}_1 \otimes \mathbf{H}_2 \\ \mathbf{G}'_1 \otimes \mathbf{H}'_2 \end{bmatrix} \cdot \text{vec}(\mathbf{Q}) = \mathbf{0}. \quad (2)$$

Then, we guess which entries of the secret matrix \mathbf{Q} are non-zero. Thanks to the structure of \mathbf{Q} , for each guess on a non-zero variable, we can simultaneously guess additional $2n - 2$ entries on the same row and column to be zero. When evaluating the variables corresponding to our guess in the system in Eq. (2), we obtain a new smaller non-homogeneous system of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$. We prove that the matrix of coefficients \mathbf{A} is not full-rank, allowing us to distinguish correct guesses from the wrong ones, with high probability, using the Rouché-Capelli Theorem: to determine whether the obtained systems accept solutions or not, we check whether $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}|\mathbf{b})$. We show that for wrong guesses $\text{rank}(\mathbf{A}) \neq \text{rank}(\mathbf{A}|\mathbf{b})$ with probability $1 - \frac{1}{q}$, which allows us to distinguish them efficiently from correct guesses.

Our algorithm consists of iterating this test on all possible n^2 guesses on the entries of \mathbf{Q} and setting the variables that did not pass the Rouché-Capelli test to zero. Our heuristic analysis shows that, for $q > 2$ and $n \geq 4$, we are able to discard enough variables so that the remaining ones can be retrieved via

Gaussian elimination, hence revealing the secret Q . The time complexity of our algorithm is indeed polynomial and consists of making two rank computations for each of the n^2 guesses, resulting in $O(n^{2+2\omega})$, for $\omega \in [2, 3]$.

We validate our theoretical results through extensive experiments and simulations performed by means of a SageMath [38] proof-of-concept implementation. All scripts are available in [13].

1.2 Related Work

Permutation Code Equivalence. The cryptanalysis of equivalence problems on linear codes started with Leon's algorithm [24], which presented a way to compute the permutation between two equivalent codes using the information provided by codewords of minimal weight, but it is unpractical for cryptographic instances. Later, Petrank and Roth [31] showed that PCE is unlikely to be NP-complete. In his seminal work [36], Sendrier introduced the Support Splitting Algorithm, which can recover the secret permutation underlying PCE in time $\tilde{O}(q^h)$, where q is the cardinality of the field and h is the dimension of the *hull* of the code, namely the intersection between the code and its dual. In addition, two more attacks on PCE with trivial hulls have been proposed [3, 33]. All these results imply that PCE is not hard when the hull is small, and this happens with high probability when the code is randomly chosen ([35] showed that in this case, the hull dimension is a small constant). Hence, PCE must be instantiated with self-dual or weakly-dual codes to be suitable in cryptography.

Linear Code Equivalence and Matrix Code Equivalence. In [37] Sendrier and Simos showed that LCE can be reduced to PCE using the closure of the code. This implied that one should be able to solve LCE using the above techniques, but, for $q \geq 5$, the closure of a code is always weakly-self dual, and the Support Splitting Algorithm becomes unfeasible. Contrary to PCE, random instances of LCE remain intractable, and hence, they can be used in the design of cryptosystems. After the publication of LESS [11], the effort for cryptanalyzing PCE and LCE increased [5, 9], which led to a refinement of the conjectured practical complexity of solving these problems. Recently, [17] showed that the LCE (as well as PCE) is equivalent to its variant which is based on canonical form of the underlying codes. In some parameter regimes, they provided the best known attacks on LCE. In summary, the known techniques are practical for particular classes of codes, while finding the permutation or the linear map leading to the equivalence seems to be still intractable for carefully generated instances. In the case of matrix codes, the equivalence problem was first studied from a cryptographic point of view in [32] and it is further cryptanalyzed in the work that introduces MEDS [16], presenting an adaptation of Leon's algorithm in the setting of matrix codes and an algebraic modelling.

Organization. We give in Sect. 2 the necessary notation and preliminaries. In Sect. 3 we give results on the sample complexity of LCE and MCE. In Sect. 4 we describe a new algorithm that solves both ILCE and 2-LCE in polynomial time, and we give the result of our experiments related to it in Sect. 5. Finally, we discuss the cryptographic implications of our work in Sect. 6.

2 Preliminaries

2.1 Notation

In this paper, we denote with \mathbb{N} , \mathbb{Z} and \mathbb{R} the sets of natural, integer and real numbers respectively. For a number $n \in \mathbb{N}$ we use $[n]$ for the set $\{1, 2, \dots, n\}$. We denote matrices with upper-case bold letters (e.g. \mathbf{A}) and vectors with lower-case bold letters (e.g. \mathbf{a}). We treat vectors as columns unless otherwise specified. Let \mathbb{F}_q denote a finite field of order q . The tensor product $(\mathbf{A} \otimes \mathbf{B}) \in \mathbb{F}_q^{mr \times ns}$ of two matrices $\mathbf{A} \in \mathbb{F}_q^{m \times n}$ and $\mathbf{B} \in \mathbb{F}_q^{r \times s}$ is defined as the Kronecker product of \mathbf{A} and \mathbf{B} .

We use $\text{GL}_n(\mathbb{F}_q)$ for the set of invertible $n \times n$ matrices with elements in \mathbb{F}_q , $\text{Perm}_n(\mathbb{F}_q)$ for the set of permutation matrices of dimension n , and $\text{Mono}_n(\mathbb{F}_q)$ for the set of $n \times n$ *monomial* matrices, i.e., that can be written as $\mathbf{M} = \mathbf{D}\mathbf{P}$, where $\mathbf{D} \in \mathbb{F}_q^{n \times n}$ is full-rank diagonal, and $\mathbf{P} \in \text{Perm}_n(\mathbb{F}_q)$. We also use \mathbf{I}_n to denote $n \times n$ identity matrix over \mathbb{F}_q .

For any matrix $\mathbf{M} \in \mathbb{F}_q^{m \times n}$, we write $\text{vec}(\mathbf{M})$ to denote the column vector of mn coefficients consisting of the concatenation of the rows of \mathbf{M} .

We assume that computing multiplication and inverse of matrices can be performed using $O(n^\omega)$ field operations for some $\omega \in [2, 3]$.² Consequently, we assume that solving a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $\mathbf{A} \in \mathbb{F}_q^{n \times n}$ and $\mathbf{b} \in \mathbb{F}_q^n$ takes time $O(n^\omega)$ field operations, and that calculating the rank (and kernel) of $\mathbf{A} \in \mathbb{F}_q^{n \times n}$ costs $O(n^\omega)$ field operations.³

The following propositions will be used in Sect. 4.

Proposition 1. *Let $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \in \mathbb{F}_q^{(k) \times (2k-1)}$ be matrices, for $k \geq 2$. Then the rank of the matrix, $\mathbf{M} \in \mathbb{F}_q^{(2k^2) \times (2k-1)^2}$, defined as*

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} \otimes \mathbf{B} \\ \mathbf{C} \otimes \mathbf{D} \end{bmatrix}$$

is strictly smaller than $2k^2$.

Proof. Given the dimension of the matrices, there exist $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\delta} \in \mathbb{F}_q^k$ non-zero vectors such that

$$[\boldsymbol{\alpha} \ \boldsymbol{\gamma}] \begin{bmatrix} \mathbf{A} \\ \mathbf{C} \end{bmatrix} = 0, \quad \text{and} \quad [\boldsymbol{\beta} \ \boldsymbol{\delta}] \begin{bmatrix} \mathbf{B} \\ \mathbf{D} \end{bmatrix} = 0.$$

² For example, in the case of the well-known Strassen's algorithm which is considered as the best algorithm for matrix multiplications for large n , one can set $\omega = \log_2(7)$.

³ If the matrix $\mathbf{A} \in \mathbb{F}_q^{r \times s}$ is rectangular, we set $n = \max\{r, s\}$ in the complexity.

Then, the vector

$$\mathbf{v} = (\boldsymbol{\alpha} \otimes \boldsymbol{\beta}, -\boldsymbol{\gamma} \otimes \boldsymbol{\delta})$$

is such that $\mathbf{v} \cdot \mathbf{M} = 0$. Indeed

$$\begin{aligned} (\boldsymbol{\alpha} \otimes \boldsymbol{\beta}, -\boldsymbol{\gamma} \otimes \boldsymbol{\delta}) \begin{bmatrix} \mathbf{A} \otimes \mathbf{B} \\ \mathbf{C} \otimes \mathbf{D} \end{bmatrix} &= (\boldsymbol{\alpha} \cdot \mathbf{A}) \otimes (\boldsymbol{\beta} \cdot \mathbf{B}) - (\boldsymbol{\gamma} \cdot \mathbf{C}) \otimes (\boldsymbol{\delta} \cdot \mathbf{D}) = \\ &(\boldsymbol{\gamma} \cdot \mathbf{C}) \otimes (\boldsymbol{\delta} \cdot \mathbf{D}) - (\boldsymbol{\gamma} \cdot \mathbf{C}) \otimes (\boldsymbol{\delta} \cdot \mathbf{D}) = 0 \end{aligned}$$

Hence, the left kernel of $\mathbf{M} \in \mathbb{F}_q^{2k^2 \times (2k-1)^2}$ is not null and it follows that $\text{rank}(\mathbf{M}) < 2k^2$. \square

2.2 Linear Codes and Equivalence Problems

An (n, k) -linear code \mathcal{C} over \mathbb{F}_q is a k -dimensional vector subspace of \mathbb{F}_q^n . We say that \mathcal{C} has length n and dimension k . The *rate* of the code is the ratio $r := \frac{k}{n}$. Unless differently specified, along this paper we consider $r \in (0, \frac{1}{2}]$.

A matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ is called a *generator matrix* of \mathcal{C} if its rows form a basis of \mathcal{C} , that is $\mathcal{C} = \{\mathbf{u}^T \mathbf{G}, \mathbf{u} \in \mathbb{F}_q^k\}$. We say that \mathbf{G} is in *systematic form* if $\mathbf{G} = (\mathbf{I}_k | \mathbf{M})$ for some $\mathbf{M} \in \mathbb{F}_q^{k \times (n-k)}$. A code that admits a generator in systematic form is called systematic code, and such generator can be obtained in polynomial-time by computing the reduced row-echelon form of a given generator. We denote this operation with $\text{SF}(\cdot)$. Moreover, the generator in systematic form gives a standard basis for the k -dimensional vector subspace of \mathbb{F}_q^n corresponding to the code. For a generator matrix \mathbf{G} , we denote $(\mathbf{G})_{-i}$ the generator matrix of the code punctured at position i , i.e., the code obtained by removing the i -th column from \mathbf{G} .

A full-rank matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ is called *parity check matrix* of \mathcal{C} if and only if $\forall \mathbf{c} \in \mathcal{C}$ it holds that $\mathbf{H}\mathbf{c} = \mathbf{0}$. Note that if $\text{SF}(\mathbf{G}) = (\mathbf{I}_k | \mathbf{M})$, for a matrix $\mathbf{M} \in \mathbb{F}_q^{k \times (n-k)}$, then the matrix $(-\mathbf{M}^\top | \mathbf{I}_{n-k})$ is a parity-check for \mathcal{C} . The parity-check matrix generates the *dual code* of \mathcal{C} , denoted with \mathcal{C}^\perp . The *hull* of a code \mathcal{C} is defined as the intersection of \mathcal{C} with its dual. A code \mathcal{C} is said *weakly self-dual* if $\mathcal{C} \subset \mathcal{C}^\perp$ and *self-dual* if $\mathcal{C} = \mathcal{C}^\perp$. In both these cases, the dimension of the hull is equal to the dimension of the code.

Due to the extended variety of namings to the Linear Code Equivalence Problem (see Table 2), and for consistency between notations in different articles, we use the acronyms from [37] and [16].

Let \mathbf{G}, \mathbf{G}' be the generator matrices of two (n, k) -linear codes $\mathcal{C}, \mathcal{C}'$. We say that \mathcal{C} and \mathcal{C}' are *equivalent* if there exist $\mathbf{S} \in \text{GL}_k(\mathbb{F}_q)$ and $\mathbf{Q} \in \text{Mono}_n(\mathbb{F}_q)$ such that $\mathbf{G}' = \mathbf{S}\mathbf{G}\mathbf{Q}$.

Definition 1 (Linear Code Equivalence (LCE) Problem). *Let $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k \times n}$ be the generator matrices of two (n, k) -linear codes $\mathcal{C}, \mathcal{C}'$, respectively. The Code Equivalence Problem is to find matrices $\mathbf{S} \in \text{GL}_k(\mathbb{F}_q)$ and $\mathbf{Q} \in \text{Mono}_n(\mathbb{F}_q)$ (if they exist) such that $\mathbf{G}' = \mathbf{S}\mathbf{G}\mathbf{Q}$.*

Table 2. Notation naming for the Linear, Permutation, and Matrix Code Equivalence Problems through the state-of-the-art.

	Permutation Code Equivalence Problem	Linear Code Equivalence Problem	Matrix Code Equivalence Problem
[23, 29, 37]	PCE	LCE	—
[34]	PEP	—	—
[2, 17, 30]	PEP	LEP	—
[7]	PEP	LEP	MCE
[16, 32]	—	—	MCE

Sometimes, in the literature, LCE is stated as in Definition 1 but with the assurance that such matrices \mathbf{S} and \mathbf{Q} establishing the equivalence between the two codes exist. Indeed, cryptographic schemes inherently guarantee the equivalence by construction. Consequently, this work explicitly addresses and incorporates this scenario.

If instead of being a monomial, the secret matrix \mathbf{Q} is a permutation matrix, then the problem is known as **Permutation Code Equivalence (PCE) Problem**.

A $(m \times r, k)$ matrix code is a subspace \mathcal{D} of dimension k of the space of $m \times r$ matrices. The following problem was introduced in [16, 32]. Two matrix codes $\mathcal{D}, \mathcal{D}'$ are *equivalent* if there exists two matrices $\mathbf{A} \in \text{GL}_m(\mathbb{F}_q)$ and $\mathbf{B} \in \text{GL}_r(\mathbb{F}_q)$ such that $\mathcal{D}' = \mathbf{A}\mathcal{D}\mathbf{B}$. In fact, [16, Lemma 1] proved that the MCE problem can be redefined in terms of the tensor product $\mathbf{A}^\top \otimes \mathbf{B}$ as described below.

Definition 2 (Matrix Code Equivalence (MCE) Problem). *Let $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k \times mr}$ be generators of two $(m \times r, k)$ -matrix codes $\mathcal{D}, \mathcal{D}'$ respectively. The Matrix Code Equivalence problem is to find (if they exist) $\mathbf{S} \in \text{GL}_k(\mathbb{F}_q)$, $\mathbf{A} \in \text{GL}_m(\mathbb{F}_q)$ and $\mathbf{B} \in \text{GL}_r(\mathbb{F}_q)$ such that $\mathbf{G}' = \mathbf{S}\mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B})$.*

Inverse Linear Code Equivalence Problem: In the context of linkable ring signatures, the following problem was initially introduced in [4].

Definition 3 (Inverse Linear Code Equivalence (ILCE) Problem). *Let $\mathbf{G}, \mathbf{G}', \mathbf{G}'' \in \mathbb{F}_q^{k \times n}$ be the generator matrices of three (n, k) -linear codes $\mathcal{C}, \mathcal{C}'$ and \mathcal{C}'' respectively. The Inverse Linear Code Equivalence Problem is to find (if they exist) $\mathbf{S} \in \text{GL}_k(\mathbb{F}_q)$ and $\mathbf{Q} \in \text{Mono}_n(\mathbb{F}_q)$ such that $\mathbf{G}' = \mathbf{S}\mathbf{G}\mathbf{Q}$ and $\mathbf{G}'' = \mathbf{S}^{-1}\mathbf{G}\mathbf{Q}^{-1}$.*

Similarly, we define the **Inverse Permutation Code Equivalence (IPCE) Problem** variant for when the secret monomial is a permutation matrix. There is also an Inverse Matrix Code Equivalence Problem variant, named IMCE and introduced in [16], that essentially replaces $\mathbf{Q} \in \text{Mono}_r(\mathbb{F}_q)$ with $\mathbf{Q} \in \text{GL}_{mr}(\mathbb{F}_q)$.

Definition 4 (Inverse Matrix Code Equivalence (IMCE) Problem). Let $\mathbf{G}, \mathbf{G}', \mathbf{G}'' \in \mathbb{F}_q^{k \times mr}$ be generators of three $(m \times r, k)$ -matrix codes $\mathcal{D}, \mathcal{D}'$ and \mathcal{D}'' respectively. The Inverse Matrix Code Equivalence problem is to find (if they exist) $\mathbf{S} \in \text{GL}_k(\mathbb{F}_q)$, $\mathbf{A} \in \text{GL}_m(\mathbb{F}_q)$ and $\mathbf{B} \in \text{GL}_r(\mathbb{F}_q)$ such that $\mathbf{G}' = \mathbf{S}\mathbf{G}\mathbf{Q}$ and $\mathbf{G}'' = \mathbf{S}^{-1}\mathbf{G}\mathbf{Q}^{-1}$ with $\mathbf{Q} = (\mathbf{A}^\top \otimes \mathbf{B}) \in \text{GL}_{mr}(\mathbb{F}_q)$.

Remark 1. In practice, one often works with generator matrices in systematic forms [2, 15]. Hence, when \mathbf{G}, \mathbf{G}' are in systematic form, we say that $\mathcal{C}, \mathcal{C}'$ are equivalent if there exists $\mathbf{Q} \in \text{Mono}_n(\mathbb{F}_q)$ such that $\mathbf{G}' = \mathbf{S}\mathbf{F}(\mathbf{G}\mathbf{Q})$. The problems LCE, PCE, MCE, and the corresponding inverse variants can all be equivalently restated with the generators in systematic form without changing the hardness of the problems. Unless differently stated, we consider these problems in their systematic form version to ease the analysis presented in this paper.

2.3 Code Equivalence Problems with Multiple Samples

In order to study the stronger cryptographic properties of the equivalence problems, we introduce some new definitions allowing an interaction with stronger adversaries. We give in Definition 5 a relaxed version of LCE where the adversary has access to multiple LCE samples for the same secret monomial \mathbf{Q} .

Definition 5 (t -LCE). Let n, k, q be positive integers such that $k < n$ and q is prime. Let $\mathbf{Q} \in \text{Mono}_n(\mathbb{F}_q)$ be a secret monomial matrix. We denote by $\mathcal{L}_{n,k,q,\mathbf{Q}}$ the probability distribution on $\mathbb{F}_q^{k \times n} \times \mathbb{F}_q^{k \times n}$ obtained by sampling $\mathbf{M} \in \mathbb{F}_q^{k \times (n-k)}$ uniformly at random, setting $\mathbf{G} = (\mathbf{I}_k | \mathbf{M}) \in \mathbb{F}_q^{k \times n}$, and returning

$$(\mathbf{G}, \mathbf{G}' = \mathbf{S}\mathbf{F}(\mathbf{G}\mathbf{Q})).$$

Given t independent samples from $\mathcal{L}_{n,k,q,\mathbf{Q}}$, the t -samples LCE problem, denoted as t -LCE, is to find \mathbf{Q} .

Informally, the distribution $\mathcal{L}_{n,k,q,\mathbf{Q}}$ samples a generator matrix \mathbf{G} (in systematic form) of a random (n, k) -linear code over \mathbb{F}_q and outputs the pair $(\mathbf{G}, \mathbf{G}')$, where \mathbf{G}' is the generator matrix (in systematic form) of another equivalent linear code, and the equivalence is established via a secret monomial matrix \mathbf{Q} . When the parameters n, k, q are clear by the context, we simplify the notation and drop the indices from the shortening of the problem, i.e., we simply write t -LCE. Also, notice that 1-LCE corresponds to LCE, so in this case only write LCE. The t -samples version problem for ILCE is as follows.

Definition 6 (t -ILCE). Let n, k, q be positive integers such that $k < n$ and q is prime. Let $\mathbf{Q} \in \text{Mono}_n(\mathbb{F}_q)$ be a secret monomial matrix. We denote by $\widehat{\mathcal{L}}_{n,k,q,\mathbf{Q}}$ the probability distribution on $\mathbb{F}_q^{k \times n} \times \mathbb{F}_q^{k \times n} \times \mathbb{F}_q^{k \times n}$ obtained by sampling $\mathbf{M} \in \mathbb{F}_q^{k \times (n-k)}$ uniformly at random, setting $\mathbf{G} = (\mathbf{I}_k | \mathbf{M}) \in \mathbb{F}_q^{k \times n}$, and returning

$$(\mathbf{G}, \mathbf{G}' = \mathbf{S}\mathbf{F}(\mathbf{G}\mathbf{Q}), \mathbf{G}'' = \mathbf{S}\mathbf{F}(\mathbf{G}\mathbf{Q}^{-1})).$$

Given t independent samples from $\widehat{\mathcal{L}}_{n,k,q,\mathbf{Q}}$, the t -samples ILCE problem, denoted as t -ILCE, is to find \mathbf{Q} .

Similarly, we call t -PCE (resp. t -IPCE) the problem of retrieving the secret matrix $\mathbf{P} \in \text{Perm}_n(\mathbb{F}_q)$ given t samples of PCE (resp. IPCE). We also refer to t -MCE (resp. t -IMCE) the problem of retrieving the secret matrices $\mathbf{A} \in \text{GL}_m(\mathbb{F}_q)$ and $\mathbf{B} \in \text{GL}_r(\mathbb{F}_q)$, from t samples of MCE (resp. IMCE).

2.4 Code Equivalences Modeled as Group Actions

A group action is a mapping of the form $\star : G \times X \rightarrow X$, where G is a group and X is a set, such that for any $g_1, g_2 \in G$ and any $x \in X$, we have $g_1 \star (g_2 \star x) = (g_1 g_2) \star x$. Cryptographic group actions are endowed with certain hardness properties, such as *one-wayness*, *weak-unpredictability* and *weak-pseudorandomness* [1].

The Linear Code Equivalence problem (Definition 1) has been modeled as a group action in [19] with the base set being $\mathbb{F}_q^{k \times n}$ and the group being $\text{GL}_k(\mathbb{F}_q) \times \text{Mono}_n(\mathbb{F}_q)$. In this work, we follow the approach similar to [7] that makes use of the systematic form. Recall that with $\text{SF}()$ we denote the computation of reduced row-echelon form. Define the following equivalence relation

$$A \simeq_{\text{SF}} B \iff \text{SF}(A) = \text{SF}(B), \quad A, B \in \mathbb{F}_q^{k \times n}.$$

Consider the base set as $X = \mathbb{F}_q^{k \times n} / \simeq_{\text{SF}}$ and the group as $G = \text{Mono}_n(\mathbb{F}_q)$. Then the group action \star is defined as

$$\star: G \times X \rightarrow X, \quad (Q, G) \mapsto Q \star G := \text{SF}(GQ).$$

Similarly, PCE and MCE are modeled as group actions following the same framework. Consequently, it follows that LCE, PCE, and MCE are instances of the so-called Vectorization Problem [18].

2-LCE, 2-PCE, and 2-MCE are special cases of the 2-GAIP defined in [7, Problem 3]. Additionally, Definition 7 describes a useful property required for building secure threshold signatures as analyzed in [7].

Definition 7 (2-weakly pseudorandom group action [7, Def. 3]). *A group action $\star: G \times X \rightarrow X$ is 2-weakly pseudorandom if there is no probabilistic polynomial time algorithm that given $(x, g \star x)$ can distinguish with non-negligible probability between (x', y') and $(x', g \star x')$ with $x', y' \in X$ sampled uniformly at random from X .*

3 Solving Code Equivalence with Multiple Instances

Recently, D’Alconzo and Di Scala [19] showed that, using representation theory, for certain group actions (G, X, \star) it is possible to recover the secret $g \in G$ from a polynomial number of samples of the form $(x_i, g \star x_i)$ for random $x_i \in X$. In the case of the group action defined in Sect. 2.4, this can be viewed as variants of the problems t -LCE, t -PCE, and t -MCE that do not use the systematic form SF . They show that these variants can be solved efficiently (with high probability)

when $t \in \text{poly}(n)$. In the case of t -LCE they showed that $t \geq nk$ samples are sufficient to recover the secret matrices \mathbf{S} and \mathbf{Q} (with high probability).

In this section, we improve the state-of-the-art by (a) showing that a significantly lower number of samples is sufficient to recover the secret matrix for the corresponding computational problem, and (b) unlike [19] our results cover even the cases when the codes are represented in the systematic form. In the rest of the paper we focus on the representation with codes in the systematic form since it leads to simpler analysis, however, we emphasize that our results extend to the general case as well since we can always compute the reduced row echelon form of the generator matrices.

In what follows, we focus our analysis on t -LCE. The main difference between t -LCE and t -MCE problems in the context of our techniques is that the secret matrix \mathbf{Q} is a monomial matrix in the case of t -LCE, whereas for t -MCE problem the secret matrix is a tensor product $(\mathbf{A}^\top \otimes \mathbf{B})$. Since we do not exploit the monomial structure of \mathbf{Q} in the following analysis of t -LCE presented in this section, our results extend in a straightforward manner to the more general case of t -MCE. Moreover, we also do not restrict the underlying linear codes to possess any specific properties or structure e.g. self-dual codes or low dimension of hull. Our strategy, analogous to [19], consists of using the available samples to construct a linear system whose unknowns are the entries of the secret matrix. If the rank of the resulting linear system is large enough, then one is able to retrieve the secret simply via Gaussian elimination in polynomial time.

Proposition 2. *Given two generator matrices $\mathbf{G} = (\mathbf{I}_k | \mathbf{M}) \in \mathbb{F}_q^{k \times n}$ and $\mathbf{G}' = \text{SF}(\mathbf{G}\mathbf{Q}) = (\mathbf{I}_k | \mathbf{M}') \in \mathbb{F}_q^{k \times n}$ of two equivalent codes for some $\mathbf{Q} \in \text{GL}_n(\mathbb{F}_q)$, we have that*

$$\left[(\mathbf{I}_k | \mathbf{M}) \otimes (-\mathbf{M}'^\top | \mathbf{I}_{n-k}) \right] \text{vec}(\mathbf{Q}) = \mathbf{0}. \quad (3)$$

Proof. This is a straightforward application of [33, Definition 1.1.3, Corollary 3.2.13, and Corollary 3.2.20] without assuming the matrix \mathbf{Q} to be a permutation, where $\mathbf{G} = (\mathbf{I}_k | \mathbf{M})$ and the parity-check matrix of the code generated by $\mathbf{G}' = (\mathbf{I}_k | \mathbf{M}')$ is given by $(-\mathbf{M}'^\top | \mathbf{I}_{n-k})$. Note that for any code generated by matrix \mathbf{G}' , by definition we know that $\mathbf{G}'\mathbf{H}^\top = \mathbf{0}$. We can therefore, obtain Eq. (3) by substituting $\mathbf{G}\mathbf{Q}$ for \mathbf{G}' and writing the resulting equation in tensor product notation. \square

Notice that Proposition 2 gives $k(n-k)$ linear equations in the n^2 variables $\text{vec}(\mathbf{Q})$ determining the entries of \mathbf{Q} .⁴ Such a linear system has the following particular structure. Let us denote the (i, j) -th entry of \mathbf{M} by $M_{i,j}$, then the homogeneous linear system of equations derived from Eq. (3) can be written as: $\mathbf{A} \cdot \text{vec}(\mathbf{Q}) = \mathbf{0}$ where \mathbf{A} is equal to

⁴ In case of LCE we restrict \mathbf{Q} to be in $\text{Mono}_n(\mathbb{F}_q)$, while for MCE we assume that $n = mr$ and $\mathbf{Q} = \mathbf{A}^\top \otimes \mathbf{B}$ for some $\mathbf{A} \in \text{GL}_m(\mathbb{F}_q)$ and $\mathbf{B} \in \text{GL}_r(\mathbb{F}_q)$.

$$\begin{bmatrix} -M'^{\top} \mathbf{I}_c & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -M_{1,1}M'^{\top} & M_{1,1}\mathbf{I}_c & \cdots & -M_{1,c}M'^{\top} & M_{1,c}\mathbf{I}_c \\ \mathbf{0} & \mathbf{0} & -M'^{\top} \mathbf{I}_c & \ddots & \vdots & -M_{2,1}M'^{\top} & M_{2,1}\mathbf{I}_c & \cdots & -M_{2,c}M'^{\top} & M_{2,c}\mathbf{I}_c \\ \vdots & \ddots & \ddots & \ddots & \ddots & \mathbf{0} & \vdots & \vdots & \cdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & -M'^{\top} \mathbf{I}_c & -M_{k,1}M'^{\top} & M_{k,1}\mathbf{I}_c & \cdots & -M_{k,c}M'^{\top} & M_{k,c}\mathbf{I}_c \end{bmatrix}$$

with $c = (n - k)$. In particular, the matrix \mathbf{A} has full (row) rank due to the presence of k identity blocks \mathbf{I}_{n-k} .

Proposition 3. *Given $t > 0$ samples from $\mathcal{L}_{n,k,q,Q}$*

$$(\mathbf{G}_i = (\mathbf{I}|\mathbf{M}_i), \mathbf{G}'_i = \text{SF}(\mathbf{G}_i\mathbf{Q}) = (\mathbf{I}|\mathbf{M}'_i)), \quad i = 1, \dots, t,$$

for a fixed secret $\mathbf{Q} \in \text{Mono}_n(\mathbb{F}_q)$, define the following matrix

$$\mathbf{A} = \begin{bmatrix} (\mathbf{I}_k|\mathbf{M}_1) \otimes (-\mathbf{M}'_1{}^{\top}|\mathbf{I}_{n-k}) \\ (\mathbf{I}_k|\mathbf{M}_2) \otimes (-\mathbf{M}'_2{}^{\top}|\mathbf{I}_{n-k}) \\ \cdots \\ (\mathbf{I}_k|\mathbf{M}_t) \otimes (-\mathbf{M}'_t{}^{\top}|\mathbf{I}_{n-k}) \end{bmatrix}. \quad (4)$$

Then $\text{rank}(\mathbf{A}) < n^2$.

Proof. Given that every LCE sample brings $k(n - k)$ rows to the matrix \mathbf{A} , there are in total $tk(n - k)$ rows and n^2 columns. If $t < \lfloor \frac{n^2}{k(n-k)} \rfloor + 1$, then the number of rows is smaller than n^2 and the rank cannot reach n^2 . Otherwise, there are always more rows than columns, hence the rank of \mathbf{A} can be at most n^2 . However, by construction we have that $\mathbf{A} \cdot \text{vec}(\mathbf{Q}) = \mathbf{0}$, hence there exists at least one linear combination of the columns of \mathbf{A} that gives the zero vector. It follows that \mathbf{A} cannot be full-rank, and so $\text{rank}(\mathbf{A}) < n^2$. \square

Studying the probability that the rank of \mathbf{A} in Eq. (4) is maximal is not an easy task. The right kernel of \mathbf{A} contains solutions of the form $\text{vec}(\mathbf{X})$ such that $\mathbf{G}'_i = \text{SF}(\mathbf{G}_i\mathbf{X})$, for $i = 1, \dots, t$, where \mathbf{X} is not necessarily monomial. In other words, such a kernel can be written as follows

$$\bigcap_{i=1, \dots, t} \{\text{vec}(\mathbf{X}) : \mathbf{G}'_i = \text{SF}(\mathbf{G}_i\mathbf{X})\}.$$

For $t > 2$, the inclusion/exclusion principle does not hold in general, and one cannot use it to estimate the dimension of such intersection of vector spaces. Nevertheless, we experimentally studied the probability that the matrix \mathbf{A} has maximal rank (i.e. $n^2 - 1$). Based on our experiments, which are reported in Sect. 3.2, we consider the following assumption.

Assumption 1. *For a given code rate r , there exist positive integers n_0, q_0 such that, for $n > n_0$, $q > q_0$, and $t \geq \lfloor \frac{1}{r(1-r)} \rfloor + 1 = \lfloor \frac{n^2}{k(n-k)} \rfloor + 1$, the matrix \mathbf{A} constructed from t random samples from $\mathcal{L}_{n,k,q,Q}$ as in Eq. (4) has rank equal to $n^2 - 1$ with non-negligible probability in n and q .*

We stress that Assumption 1 is meant to cover all cryptographically interesting cases. Indeed, experimentally, we observed that such an assumption does not hold true either for codes of very short length and small field or for very small rate r , which are not of cryptographic interest. For example, for $r = \frac{1}{2}$, Assumption 1 seems to hold for $n_0 = 8$ and $q_0 = 3$. Under the hypothesis of Assumption 1, we give the sample complexity of LCE in Lemma 1.

Lemma 1. *For $t \geq \left\lfloor \frac{n^2}{k(n-k)} \right\rfloor + 1$ and under Assumption 1, t -LCE is solvable with non-negligible probability in time $O(n^{2\omega})$.*

Proof. Construct the matrix \mathbf{A} from t LCE samples sharing the same secret \mathbf{Q} as in Proposition 3. Following Assumption 1, the right kernel of \mathbf{A} has dimension equal to 1. The generator of such kernel, which can be found via Gaussian elimination, must be (a multiple of) $\text{vec}(\mathbf{Q})$ by construction, and so, a solution for each of the t LCE instances. \square

Notice that Lemma 1 also applies to PCE as this can be seen as a special case of LCE. We assume an analogue of Assumption 1 for the case of MCE by setting $n := mr$ in the following corollary.

Corollary 1. *For $t \geq \left\lfloor \frac{m^2 r^2}{k(mr-k)} \right\rfloor + 1$, t -MCE is solvable with non-negligible probability in time $O((mr)^{2\omega})$.*

In the parameter setting used LESS [2], i.e. $k = n/2$, Lemma 1 says that a constant number of $t = 5$ samples are enough, for any n , to recover the secret monomial. Similarly, for the parameter setting used in MEDS [15], i.e. $k = r = m$, Corollary 1 says that $t = \left\lfloor \frac{k^2}{k-1} \right\rfloor \approx k$ samples are enough to recover the secret matrix. We stress that this has no implication on the security of such protocols because, in both protocols setting, only one sample is provided.

We verified experimentally in SageMath the correctness of Lemma 1 and Corollary 1, and the scripts are available at [13].

3.1 Implications to ILCE and IMCE

Consider the following ILCE instance

$$(\mathbf{G}, \mathbf{G}' = \text{SF}(\mathbf{G}\mathbf{Q}), \mathbf{G}'' = \text{SF}(\mathbf{G}\mathbf{Q}^{-1})).$$

By multiplying \mathbf{Q} to the right in the equation at the right-most entry, one gets

$$(\mathbf{G}, \mathbf{G}' = \text{SF}(\mathbf{G}\mathbf{Q}), \mathbf{G} = \text{SF}(\mathbf{G}''\mathbf{Q})),$$

that is *almost* a 2-LCE sample. Indeed, these two resulting LCE samples do not come both from $\mathcal{L}_{n,k,q,\mathbf{Q}}$, but are instead related to each other by the matrix \mathbf{G} appearing twice, even if on different positions. Nevertheless, we argue below

that, for the sake of our analysis, t -ILCE with the above transformation behaves as $2t$ -LCE.

Given $t > 0$ random ILCE samples

$$(\mathbf{G}_i = (\mathbf{I}_k | \mathbf{M}_i), \mathbf{G}'_i = (\mathbf{I}_k | \mathbf{M}'_i), \mathbf{G}''_i = (\mathbf{I}_k | \mathbf{M}''_i)), \quad \text{for } i = 1, \dots, t,$$

consider the matrix

$$\mathbf{A}' = \begin{bmatrix} (\mathbf{I}_k | \mathbf{M}_1) \otimes (-\mathbf{M}_1^\top | \mathbf{I}_{n-k}) \\ (\mathbf{I}_k | \mathbf{M}'_1) \otimes (-\mathbf{M}'_1^\top | \mathbf{I}_{n-k}) \\ \dots \\ (\mathbf{I}_k | \mathbf{M}_t) \otimes (-\mathbf{M}_t^\top | \mathbf{I}_{n-k}) \\ (\mathbf{I}_k | \mathbf{M}''_t) \otimes (-\mathbf{M}''_t^\top | \mathbf{I}_{n-k}) \end{bmatrix}. \quad (5)$$

By construction, we have that $\mathbf{A}' \cdot \text{vec}(\mathbf{Q}) = \mathbf{0}$. Hence, for analogous arguments as in the proof of Proposition 3, the matrix \mathbf{A}' has rank always smaller than n^2 . Looking at matrix \mathbf{A}' , even if each matrix \mathbf{M}_i appears in two row-blocks, they are in different columns and they do not seem to bring any evident linear dependence. In addition, we observed experimentally that, for $t = \lfloor \frac{n^2}{2k(n-k)} \rfloor + 1$, the probability of $\text{rank}(\mathbf{A}') \neq n^2 - 1$ is analogous to the case of $2t$ -LCE (the results of our experiments are reported in Sect. 3.2). On the basis of the above considerations, we consider Assumption 2 in order to give the sample complexity of ILCE in Lemma 2.

Assumption 2. *For a given code rate r , there exist positive integers n_0, q_0 such that, for $n > n_0, q > q_0$, and $t \geq \lfloor \frac{1}{2r(1-r)} \rfloor + 1 = \lfloor \frac{n^2}{2k(n-k)} \rfloor + 1$, the matrix \mathbf{A}' constructed from t random samples from $\mathcal{L}_{n,k,q,Q}$ as in Eq. (5) has rank equal to $n^2 - 1$ with non-negligible probability in n and q .*

Lemma 2. *Under Assumption 2, for $t \geq \lfloor \frac{n^2}{2k(n-k)} \rfloor + 1$, t -ILCE is solvable with non-negligible probability in time $O(n^{2\omega})$.*

Proof. Analogous to the proof of Lemma 1. □

Similarly to what is done for MCE, under an analogous assumption to Assumption 2 but for IMCE, we have the following corollary.

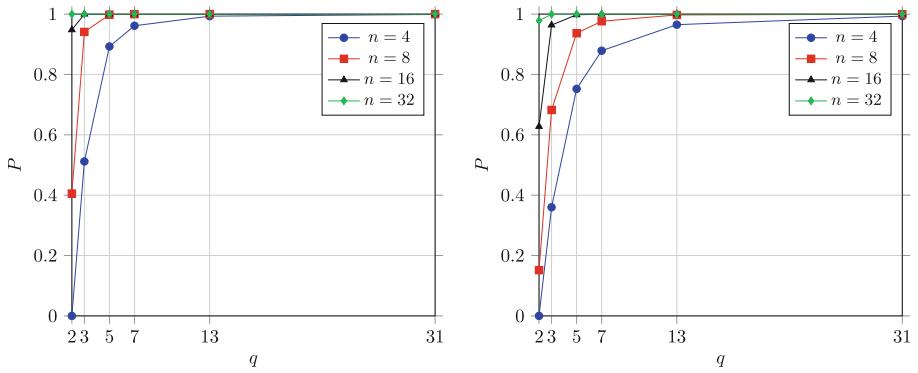
Corollary 2. *For $t \geq \lfloor \frac{m^2 r^2}{2k(mr-k)} \rfloor + 1$, t -IMCE is solvable with non-negligible probability in time $O((mr)^{2\omega})$.*

Similarly, as above, we verified experimentally in SageMath the correctness of Lemma 2 and Corollary 2.

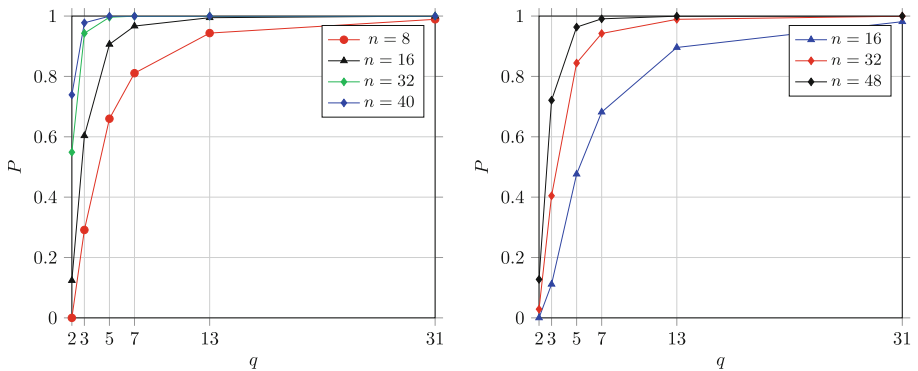
3.2 Experimental Validation of Assumptions

In this section, we report the results from our experiments for testing whether Assumption 1 and Assumption 2 hold in practice.

Experiments on Assumption 1. Our experiment consists of testing, for a range of n, q and code rate $r = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$, how many matrices constructed as in Eq. (4) have rank equal to $n^2 - 1$. For each rate, we choose $t = \lfloor \frac{1}{r(1-r)} \rfloor + 1$ and run 10000 trials and report in Fig. 1 the fraction of how many trials presented the desired maximal rank. One can see that the measured probability of this event to happen quickly goes to 1 when either or both q and n increase. In addition, one can notice that when the code rate r is close to either 0 or 1, the probability of reaching the maximal rank is lower. Tests for rate $r > \frac{1}{2}$ gave symmetrical results as for $r < \frac{1}{2}$. In overall, our experimental results support Assumption 1.



(a) Code dimension set to $k = \frac{n}{2}$ (left) and $k = \frac{n}{4}$ (right).



(b) Code dimension set to $k = \frac{n}{8}$ (left) and $k = \frac{n}{16}$ (right).

Fig. 1. The plots report the measured success rate P over 10000 trials that a matrix \mathbf{A} from Eq. (4), constructed from $t = \lfloor \frac{n^2}{k(n-k)} \rfloor + 1$ random LCE samples with parameters n, k and q , has rank equal to $n^2 - 1$. Each plots shows the results for different values of n, q and a code rate r equal to $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$ and $\frac{1}{16}$.

Experiments on Assumption 2. Under analogous setting of the above experiment, we test whether the matrix \mathbf{A}' from Eq.(5) has the desired rank $n^2 - 1$, for $t = \lfloor \frac{1}{2r(1-r)} \rfloor + 1$. The results reported in Fig. 2 support Assumption 2.

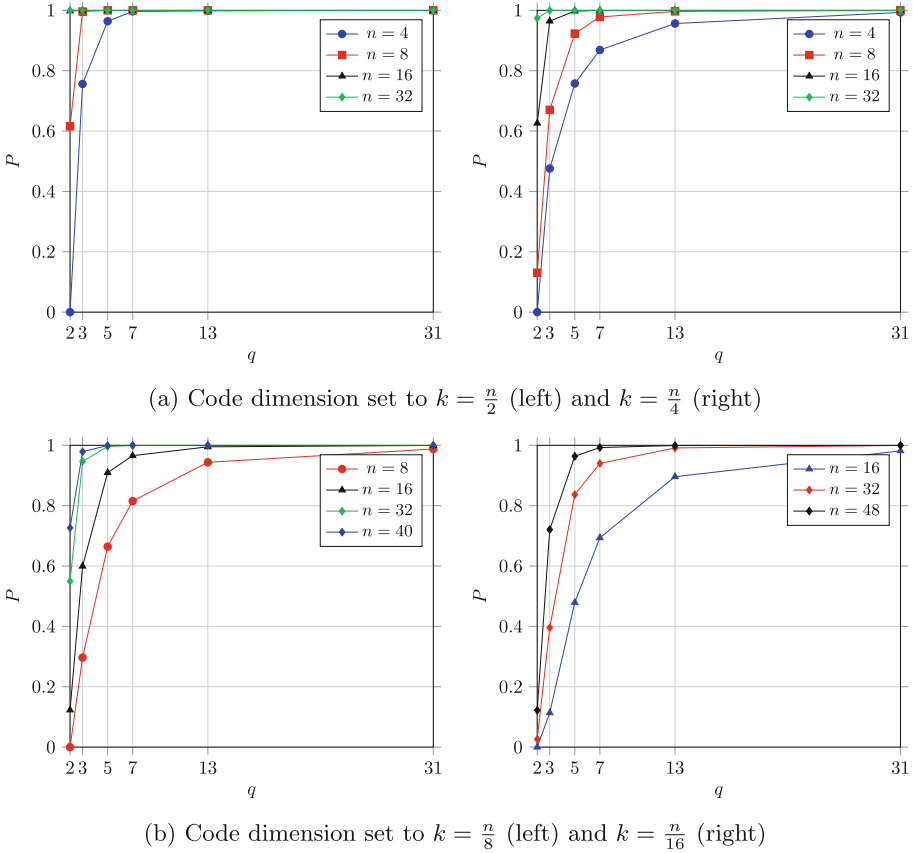


Fig. 2. The plots report the measured success rate P over 10000 trials that a matrix \mathbf{A} from Eq. (4), constructed from $t = \lfloor \frac{n^2}{2k(n-k)} \rfloor + 1$ random lLCE samples with parameters n, k and q , has rank equal to $n^2 - 1$. Each plots shows the results for different values of n, q and a code rate r equal $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$ and $\frac{1}{16}$.

3.3 Solving LCE when rank(\mathbf{A}) is not maximal

In practice, one can solve t -LCE also when the right rank of the constructed matrix \mathbf{A} from Eq. (4) is not strictly maximal (i.e. $\text{rank}(\mathbf{A}) = n^2 - 1$) as assumed in Assumption 1. Indeed, we show here that if the right kernel of \mathbf{A} has dimension

between 2 and n , then the monomial solution (or a multiple of it) must be an element of a specific set of generators with high probability, and this set can be computed in polynomial time.

Let us assume that the right kernel of \mathbf{A} has dimension n and let $\mathbf{g}_1, \dots, \mathbf{g}_n \in \mathbb{F}_q^{n^2}$ be its generators (smaller rank cases are analogous). With high probability, there exists a basis transformation that makes $\mathbf{g}_1, \dots, \mathbf{g}_n$ a standard basis, that is, where

$$\begin{aligned} \mathbf{g}_1 &= \&(1, 0, \dots, 0, \overbrace{*, \dots, *}^n) \\ \mathbf{g}_2 &= \&(0, 1, \dots, 0, *, \dots, *) \\ &\quad \&\dots \\ \mathbf{g}_n &= \&(0, \dots, 0, 1, *, \dots, *). \end{aligned}$$

Let us assume that $\text{vec}(\mathbf{Q}) \neq \alpha \mathbf{g}_i$, for any $\alpha \in \mathbb{F}_q^*$ and $i = 1, \dots, n$. Then it means that $\text{vec}(\mathbf{Q})$ must be a linear combination of $\mathbf{g}_1, \dots, \mathbf{g}_n$. However, because of the monomial structure of \mathbf{Q} , $\text{vec}(\mathbf{Q})$ has only one non-zero entry in its first n entries. It follows that $\text{vec}(\mathbf{Q})$ cannot be a combination of two or more \mathbf{g}_i as this would generate more than one non-zero entry in these first n positions. Hence, $\text{vec}(\mathbf{Q})$ must be one of the elements of the standard basis of the kernel or a multiple of it.

Thanks to this observation, we were able in practice to solve the cases in which $n^2 - n \leq \text{rank}(\mathbf{A}) \leq n^2 - 1$, and, in particular, for $k = \frac{n}{2}$, we could find the secret monomial with only 4 random LCE samples, which decreases by 1 the sample complexity given by Lemma 1.

4 Further Improvements by Exploiting the Monomial Matrix Structure

In this section, we exploit the structure of the secret matrix in LCE and ILCE to further reduce the number of samples necessary to retrieve the secret monomial. Specifically, we show how to solve, in polynomial time, 2-LCE and ILCE for code rate $\frac{1}{2}$. The approach presented below builds upon the algorithm by Saeed for PCE [33, Sec. 3.7]. As in Sect. 3, we consider the generators of the codes in systematic form to ease our analysis.

4.1 Solving 2-LCE for $k = n/2$ in polynomial-time

In this section, we introduce a new algorithm that solves 2-LCE in polynomial time for codes of rate $\frac{1}{2}$. Consider a secret matrix $\mathbf{Q} \in \text{Mono}_n(\mathbb{F}_q)$ and the following two LCE instances:

$$\begin{aligned} \left(\mathbf{G}_1 = (\mathbf{I}_k | \mathbf{M}), \&\mathbf{G}'_1 = \text{SF}(\mathbf{G}_1 \mathbf{Q}) = (\mathbf{I}_k | \mathbf{M}') \right), \\ \left(\mathbf{G}_2 = (\mathbf{I}_k | \mathbf{N}), \&\mathbf{G}'_2 = \text{SF}(\mathbf{G}_2 \mathbf{Q}) = (\mathbf{I}_k | \mathbf{N}') \right). \end{aligned} \tag{6}$$

for $k = n/2$. We apply Proposition 2 to each instance and write the following homogeneous linear system

$$S : \overbrace{\begin{bmatrix} (\mathbf{I}_k | \mathbf{M}) \otimes (-\mathbf{M}'^\top | \mathbf{I}_{n-k}) \\ (\mathbf{I}_k | \mathbf{N}) \otimes (-\mathbf{N}'^\top | \mathbf{I}_{n-k}) \end{bmatrix}}^{\mathbf{A}} \text{vec}(\mathbf{Q}) = \begin{bmatrix} \mathbf{0} \end{bmatrix}. \quad (7)$$

The following proposition gives a sufficient and necessary condition for \mathbf{A} to be full-rank.

Proposition 4. *Consider two LCE instances as in Eq. (6), for $k = \frac{n}{2}$. Then the matrix \mathbf{A} defined in Eq. (7) is such that $\text{rank}(\mathbf{A}) < 2k(n-k)$ if and only if $\text{rank}(\mathbf{M} - \mathbf{N}) < k$.*

Before giving the proof for Proposition 4, we need to prove the following proposition.

Proposition 5. *Under the same setting of Proposition 4, we have that*

$$\text{rank}(\mathbf{M}' - \mathbf{N}') < k \iff \text{rank}(\mathbf{M} - \mathbf{N}) < k.$$

Proof. First, we prove that $\text{rank}(\mathbf{M}' - \mathbf{N}') < k \Rightarrow \text{rank}(\mathbf{M} - \mathbf{N}) < k$. One has that

$$\mathbf{G}'_1 - \mathbf{G}'_2 = (\mathbf{I}_k | \mathbf{M}') - (\mathbf{I}_k | \mathbf{N}') = \mathbf{X}(\mathbf{I}_k | \mathbf{M})\mathbf{Q} - \mathbf{Y}(\mathbf{I}_k | \mathbf{N})\mathbf{Q}$$

for some invertible $\mathbf{X}, \mathbf{Y} \in \mathbb{F}_q^{k \times k}$. Then

$$(\mathbf{0} | \mathbf{M}' - \mathbf{N}')\mathbf{Q}^{-1} = (\mathbf{X}(\mathbf{I}_k | \mathbf{M}) - \mathbf{Y}(\mathbf{I}_k | \mathbf{N})) = (\mathbf{X} | \mathbf{Y}) \begin{bmatrix} \mathbf{I}_k & \mathbf{M} \\ -\mathbf{I}_k & -\mathbf{N} \end{bmatrix}.$$

For any matrix \mathbf{Z} let $\ker_L(\mathbf{Z})$ be its left kernel. Let $\mathbf{w}^\top \in \ker_L(\mathbf{M}' - \mathbf{N}')$, then

$$\mathbf{0} = \mathbf{w}^\top \cdot (\mathbf{0} | \mathbf{M}' - \mathbf{N}')\mathbf{Q}^{-1} = \mathbf{w}^\top \cdot (\mathbf{X} | \mathbf{Y}) \begin{bmatrix} \mathbf{I}_k & \mathbf{M} \\ -\mathbf{I}_k & -\mathbf{N} \end{bmatrix}.$$

It follows that $\mathbf{w}'^\top = \mathbf{w}^\top \cdot (\mathbf{X} | \mathbf{Y}) \in \ker_L \left(\begin{bmatrix} \mathbf{I}_k & \mathbf{M} \\ -\mathbf{I}_k & -\mathbf{N} \end{bmatrix} \right)$. Notice that $\mathbf{w}' \in \mathbb{F}_q^n$ must be of the form $\mathbf{w}' = (\mathbf{v}, \mathbf{v})$, with $\mathbf{v} \neq \mathbf{0} \in \mathbb{F}_q^k$ and $\mathbf{v}^\top \in \ker_L(\mathbf{M} - \mathbf{N})$. Hence we have that $\text{rank}(\mathbf{M} - \mathbf{N}) < k$. The other implication $\text{rank}(\mathbf{M} - \mathbf{N}) < k \Rightarrow \text{rank}(\mathbf{M}' - \mathbf{N}') < k$ follows by using analogous arguments as above. \square

We can now give the proof of Proposition 4.

Proof (Proposition 4). First, we prove that $\text{rank}(\mathbf{M} - \mathbf{N}) < k \Rightarrow \text{rank}(\mathbf{A}) < 2k(n-k)$. Notice that since \mathbf{A} has n^2 columns and $2k(n-k) = \frac{n^2}{2}$ rows, $\text{rank}(\mathbf{A})$ can be at most equal to $2k(n-k)$. Let $\mathbf{v}^\top \neq \mathbf{0} \in \ker_L(\mathbf{M} - \mathbf{N})$, then there exists $\mathbf{w}^\top \neq \mathbf{0} \in \ker_L(\mathbf{M}' - \mathbf{N}')$ from Proposition 5. Then we have that

$$(\mathbf{v}^\top | -\mathbf{v}^\top) \begin{bmatrix} (\mathbf{I}_k | \mathbf{M}) \\ (\mathbf{I}_k | \mathbf{N}) \end{bmatrix} = \mathbf{0} \quad \text{and} \quad (\mathbf{w}^\top | -\mathbf{w}^\top) \begin{bmatrix} (-\mathbf{M}'^\top | \mathbf{I}_{n-k}) \\ (-\mathbf{N}'^\top | \mathbf{I}_{n-k}) \end{bmatrix} = \mathbf{0}.$$

It follows that

$$(\mathbf{v}^\top \otimes \mathbf{w}^\top | -\mathbf{v}^\top \otimes \mathbf{w}^\top) \begin{bmatrix} (\mathbf{I}_k | \mathbf{M}) \otimes (-\mathbf{M}'^\top | \mathbf{I}_{n-k}) \\ (\mathbf{I}_k | \mathbf{N}) \otimes (-\mathbf{N}'^\top | \mathbf{I}_{n-k}) \end{bmatrix} = \mathbf{0}.$$

Hence, $(\mathbf{v}^\top \otimes \mathbf{w}^\top | -\mathbf{v}^\top \otimes \mathbf{w}^\top) \neq \mathbf{0} \in \ker_L(\mathbf{A})$ and so $\text{rank}(\mathbf{A}) < 2k(n-k)$.

We prove now that $\text{rank}(\mathbf{A}) < 2k(n-k) \Rightarrow \text{rank}(\mathbf{M} - \mathbf{N}) < k$. Let $\mathbf{s}^\top = (\mathbf{s}_1^\top, \mathbf{s}_2^\top) \neq \mathbf{0} \in \ker_L(\mathbf{A})$. If we restrict the multiplication $\mathbf{s}^\top \mathbf{A} = \mathbf{0}$ to the first $2k$ columns of \mathbf{A} , we get the following equation

$$(\mathbf{s}_1^\top | \mathbf{s}_2^\top) \begin{bmatrix} -\mathbf{M}'^\top & \mathbf{I}_{n-k} \\ \mathbf{0} & \mathbf{0} \\ -\mathbf{N}'^\top & \mathbf{I}_{n-k} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} = \mathbf{0}.$$

Let $\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2 \in \mathbb{F}_q^k$ be the vectors of the first k entries of \mathbf{s}_1 and \mathbf{s}_2 respectively. Then we have that

$$(\bar{\mathbf{s}}_1^\top | \bar{\mathbf{s}}_2^\top) \begin{bmatrix} -\mathbf{M}'^\top & \mathbf{I}_{n-k} \\ -\mathbf{N}'^\top & \mathbf{I}_{n-k} \end{bmatrix} = \mathbf{0}.$$

It follows that $\bar{\mathbf{s}}_1^\top + \bar{\mathbf{s}}_2^\top = \mathbf{0}$, therefore $-\bar{\mathbf{s}}_1^\top = \bar{\mathbf{s}}_2^\top$ and that $\bar{\mathbf{s}}_1^\top \in \ker_L(\mathbf{M}'^\top - \mathbf{N}'^\top)$ and, for $k = \frac{n}{2}$, $\text{rank}(\mathbf{M}' - \mathbf{N}') < k$. From Proposition 5 we conclude that $\text{rank}(\mathbf{M} - \mathbf{N}) < k$. \square

Description of the Algorithm. The main idea of our algorithm is to infer information about the secret monomial matrix \mathbf{Q} by guessing the position of the non-zero entry in each row and checking whether the resulting reduced system admits solutions. More specifically, we iteratively guess the entries of \mathbf{Q} to be non-zero. Each guess consists of evaluating the variable corresponding to the (i, j) -th entry of \mathbf{Q} to be equal to 1. Thanks to the monomial structure of \mathbf{Q} , this results in guessing a total of $2n - 1$ variables simultaneously, since all the remaining variables in the i -th row and in the j -th column must be equal to 0. Then, we either retain or discard the guess depending on whether the reduced linear system obtained from such a guess admits any solution(s) or not.

We now explain why such a guess on the correct non-zero position of \mathbf{Q} is still useful even if $\mathbf{Q}(i, j) \neq 1$. Recall that $\mathbf{Q} = \mathbf{P}\mathbf{D}$, where \mathbf{P} is a permutation matrix in $\text{Perm}_n(\mathbb{F}_q)$ and \mathbf{D} is a diagonal matrix in $\text{GL}_n(\mathbb{F}_q)$. Let $d_i \in \mathbb{F}_q^*$ be the i -th diagonal entry of \mathbf{D} . Then $\mathbf{R}_i = d_i^{-1}\mathbf{Q}$ satisfies $\mathbf{G}'_1 = \text{SF}(\mathbf{G}_1\mathbf{R}_i)$ and $\mathbf{G}'_2 = \text{SF}(\mathbf{G}_2\mathbf{R}_i)$ for each $i \in \{1, \dots, n\}$. In other words, this guess restricts the set of possible solutions to include a specific multiple \mathbf{R}_i of \mathbf{Q} that has 1 in its i -th non-zero entry (due to scaling by d_i) which also serves as a solution to the given 2-LCE instance. Therefore, such an evaluation on the non-zero entry remains valid.

We give here a characterization of the linear system obtained by guessing a single position. Setting $\mathbf{Q}(i, j) = 1$ and $\mathbf{Q}(i, \mu), \mathbf{Q}(\eta, j) = 0$, for $\mu \in \{1 \dots n\} \setminus \{j\}$ and $\eta \in \{1 \dots n\} \setminus \{i\}$, results in removing the corresponding $2n - 1$ columns of \mathbf{A} from the linear system \mathbf{S} in Eq. (7). This operation produces the linear system

$$\mathbf{S}_{i,j} : \mathbf{A}_{ij} \cdot \text{vec}(\mathbf{Q}') = \mathbf{b}_{ij} \quad (8)$$

in dimension $2k(n - k) \times (n - 1)^2$, where

$$\mathbf{A}_{ij} = \begin{bmatrix} (\mathbf{I}_k | \mathbf{M})_{-i} \otimes (-\mathbf{M}'^\top | \mathbf{I}_{n-k})_{-j} \\ (\mathbf{I}_k | \mathbf{N})_{-i} \otimes (-\mathbf{N}'^\top | \mathbf{I}_{n-k})_{-j} \end{bmatrix},$$

$$-\mathbf{b}_{ij} = \begin{bmatrix} (\mathbf{I}_k | \mathbf{M})_i \otimes (-\mathbf{M}'^\top | \mathbf{I}_{n-k})_j \\ (\mathbf{I}_k | \mathbf{N})_i \otimes (-\mathbf{N}'^\top | \mathbf{I}_{n-k})_j \end{bmatrix},$$

and \mathbf{Q}' is the $(n - 1) \times (n - 1)$ matrix obtained by removing the i -th row and j -th column from \mathbf{Q} . In other words, we obtain a new non-homogeneous linear system given by the tensor product of \mathbf{G} punctured at position i and \mathbf{H}' (parity check matrix of \mathbf{G}') punctured at position j . Notice that the vector of the constant terms \mathbf{b}_{ij} corresponds to the $(n(i - 1) + j)$ -th column of the original matrix \mathbf{A} , i.e., the one corresponding to the variable $\mathbf{Q}(i, j)$ that is guessed to be non-zero.

On each guess, we use the following test to accept or reject a guess.

Test 1. *For the guess on the (i, j) -th entry of \mathbf{Q} to be non-zero, construct a reduced system \mathbf{S}_{ij} from \mathbf{S} (as in Eq. (8)) with $(n - 1)^2$ variables by setting $\mathbf{Q}(i, j) = 1$ and $\mathbf{Q}(i, \mu), \mathbf{Q}(\eta, j) = 0$, for $\mu \in \{1 \dots n\} \setminus \{j\}$ and $\eta \in \{1 \dots n\} \setminus \{i\}$. Accept the guess if the system \mathbf{S}_{ij} accepts at least one solution, reject otherwise.*

We use Rouché-Capelli Theorem to check whether \mathbf{S}_{ij} accepts solutions or not. Indeed, the system \mathbf{S}_{ij} accepts solutions if and only if $\text{rank}(\mathbf{A}_{ij}) = \text{rank}(\mathbf{A}_{ij} | \mathbf{b}_{ij})$. When a guess is rejected, this means that no solution in \mathbf{S} exists with $\mathbf{Q}(i, j) \neq 0$. Hence, the variable corresponding to $\mathbf{Q}(i, j)$ in \mathbf{S} is set to zero. If enough variables are set to zero after the guessing procedure, i.e. the system becomes (over)determined, and we can retrieve the remaining ones using Gaussian elimination. The whole strategy is outlined in Algorithm 1.

Algorithm 1. Solving 2-LCE**Input:** A 2-LCE instance as in Eq. (6)**Output:** A monomial matrix \mathbf{R} , solution to Eq. (6) or \perp

```

1: Construct the linear system  $\mathbf{S}$  given by Eq. (7)
2: Set  $g = [g_1, \dots, g_n]$  such that  $g_i$  is an empty list
3: for  $i := 1$  to  $n$  do ▷ loop over rows
4:   for  $j := 1$  to  $n$  do ▷ loop over columns
5:     if Test 1 passes then
6:       Append  $j$  to the list  $g_i$ 
7:     end if
8:   end for
9: end for
10: Construct the linear system  $\mathbf{S}_{\text{red}}$  obtained by substituting  $\mathbf{Q}(i, j) = 0$  in  $\mathbf{S}$  for each
     $i := 1, \dots, n$  and  $j \notin g_i$ 
11: if  $\mathbf{S}_{\text{red}}$  is underdetermined then
12:   Return  $\perp$ 
13: end if
14: Compute a solution matrix  $\mathbf{R}$  of the linear system  $\mathbf{S}_{\text{red}}$ 
15: Return  $\mathbf{R}$ 

```

Notice that, when Algorithm 1 succeeds, it returns an equivalent solution (a scalar multiple of) to the original secret matrix \mathbf{Q} .

Heuristic Analysis of Algorithm 1. First of all, notice that Test 1 always accepts a correct guess since, in this case, \mathbf{S}_{ij} accepts solutions by construction. On the other hand, Test 1 may or may not accept a wrong guess. Thus, we begin our analysis by estimating the probability that Test 1 accepts a wrong guess.

Proposition 4 gives the condition for which the matrix of coefficients \mathbf{A} in Eq. (7) is full rank. In particular, given that \mathbf{M}, \mathbf{N} are sampled uniformly at random, we know that

$$Pr\left(\text{rank}(\mathbf{A}) = \frac{n^2}{2}\right) = Pr(\text{rank}(\mathbf{M} - \mathbf{N}) = k) = 1 - \frac{1}{q}. \quad (9)$$

On the other hand, Proposition 1 applied to the matrix of coefficients \mathbf{A}_{ij} of the reduced system \mathbf{S}_{ij} (Eq. (8)) tells us that $\text{rank}(\mathbf{A}_{ij}) = \frac{n^2}{2} - d$, for some $d > 0$. Using Rouché-Capelli Theorem to check whether \mathbf{S}_{ij} admits solutions or not, implies that the guess (i, j) passes Test 1 if and only if $\text{rank}(\mathbf{A}_{ij}|\mathbf{b}_{ij}) = \text{rank}(\mathbf{A}_{ij}) = \frac{n^2}{2} - d$.

Let X be the left kernel of \mathbf{A}_{ij} . The dimension of X is $\frac{n^2}{2} - \text{rank}(\mathbf{A}_{ij}) = d$, and let $\mathbf{B}_X \in \mathbb{F}_q^{d \times \frac{n^2}{2}}$ be its generator matrix. Similarly, Let Y be the left kernel of \mathbf{b}_{ij} of dimension $\frac{n^2}{2} - \text{rank}(\mathbf{b}_{ij}) = \frac{n^2}{2} - 1$ and let $\mathbf{B}_Y \in \mathbb{F}_q^{(\frac{n^2}{2}-1) \times \frac{n^2}{2}}$ be its generator matrix. Then, we have that

$$\text{rank}(\mathbf{A}_{ij}|\mathbf{b}_{ij}) = \text{rank}(\mathbf{A}_{ij}) \iff X \subset Y \iff \text{rank}(\mathbf{B}_Y) = \text{rank}\left(\begin{bmatrix} \mathbf{B}_Y \\ \mathbf{B}_X \end{bmatrix}\right).$$

Heuristically, we model \mathbf{B}_X and \mathbf{B}_Y as random matrices, and the probability that all rows of \mathbf{B}_X are linearly dependant from the rows of \mathbf{B}_Y is approximately equal to $\frac{1}{q^d}$. Therefore, the expected probability that a wrong guess passes Test 1 is $\frac{1}{q^d}$.

Let us now estimate the expected number of variables that will pass Test 1, i.e., the number of variables of the system S_{red} in Algorithm 1. Here, we consider the most probable scenario of $d = 1$ (that is also the worst case scenario, since for $d > 1$ Test 1 accepts wrong guesses with lower probability). In total, there are n correct guesses (one for each row) that will always pass Test 1, and the remaining $n^2 - n$ incorrect guesses will pass with probability $\frac{1}{q}$. The expected number of survival variables is

$$N = n + (n^2 - n)\frac{1}{q}. \quad (10)$$

We have that the resulting system is (over)determined when $N \leq \frac{n^2}{2}$, and this is true when

$$q \geq \frac{2(n-1)}{n-2}. \quad (11)$$

Notice that, for $q > 2$ and $n \geq 4$, Eq. (11) is always satisfied. Hence, when the parameters satisfy Eq. (11), the condition that determines the success of Algorithm 1 is that $\text{rank}(\mathbf{A}) = \frac{n^2}{2}$, which happens with probability $1 - \frac{1}{q}$ (see Eq. (9)).

Complexity. The computational cost of checking $\text{rank}(\mathbf{A}_{ij}|\mathbf{b}_{ij}) = \text{rank}(\mathbf{A}_{ij})$ is $O(n^{2\omega})$, for $\omega \in [2, 3]$. This computation must be repeated for n^2 guesses, giving a computational complexity of

$$O(n^{2+2\omega})$$

field operations. The memory complexity is of $O(n^4)$ field elements.

In order to check the correctness of Algorithm 1 and of the proposed analysis, we perform extensive experiments in SageMath up to code length $n = 128$ as discussed in Sect. 5.1.

4.2 Solving ILCE for $k = n/2$ in polynomial-time

Consider an ILCE instance

$$(\mathbf{G} = (\mathbf{I}_k|\mathbf{M}), \mathbf{G}' = (\mathbf{I}_k|\mathbf{M}'), \mathbf{G}'' = (\mathbf{I}_k|\mathbf{M}'')).$$

Following the same reasoning as in Sect. 3.1, we obtain a system which is *almost* same as the one obtained from a 2-LCE instance. For Algorithm 1 to work, we need first to check that the following matrix

$$\mathbf{A}' = \begin{bmatrix} (\mathbf{I}_k|\mathbf{M}) \otimes (-\mathbf{M}'^\top|\mathbf{I}_{n-k}) \\ (\mathbf{I}_k|\mathbf{M}'') \otimes (-\mathbf{M}^\top|\mathbf{I}_{n-k}) \end{bmatrix}. \quad (12)$$

is full rank. According to Proposition 4, \mathbf{A}' is full-rank if and only if $\ker_{\mathbb{L}}(\mathbf{M} - \mathbf{M}'')$ is trivial. Heuristically, given that \mathbf{M} is random and modelling \mathbf{M}'' as also random, \mathbf{A}' is full rank with probability $1 - \frac{1}{q}$. Then, guessing variables of the system $\mathbf{A}' \cdot \text{vec}(\mathbf{Q}) = \mathbf{0}$ produces a system analogous to Eq. (8), where Proposition 1 naturally applies to \mathbf{A}' . Consequently, the requirements for Algorithm 1 are met, allowing us to solve ILCE in polynomial time.

Our experiments, reported in Sect. 5.2, show that Algorithm 1 solves 2-LCE and ILCE with analogous success probability.

4.3 Solving 2-PCE and IPCE for self-dual codes

Since PCE is a special case of LCE, the above results also apply to IPCE and 2-PCE for random codes. However, in this case, it is already known that PCE can be solved in polynomial time using the Support Splitting Algorithm [36]. Unfortunately, for the case of self-dual codes, this approach has exponential time complexity.

We argue that the hull of the code does not play a role in our algorithm. First, notice that when building the system in Eq. (7), the code and its dual are never used simultaneously (instead, we use the dual of an equivalent code). Specifically, given two PCE instances

$$(\mathbf{G}_1, \mathbf{G}'_1) \quad (\mathbf{G}_2, \mathbf{G}'_2),$$

with secret permutation matrix \mathbf{P} , we construct the system (notice that in this case, \mathbf{G}'_i is the dual of itself)

$$\begin{bmatrix} \mathbf{G}_1 \otimes \mathbf{G}'_1 \\ \mathbf{G}_2 \otimes \mathbf{G}'_2 \end{bmatrix} \text{vec}(\mathbf{P}) = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \quad (13)$$

Second, the key factor that makes the Rouché-Capelli test work is that the matrix of coefficients after puncturing \mathbf{A}_{ij} must not be full-rank, and this is true via Proposition 1 regardless of the dimension of the hull. Finally, our experiments, reported in Sect. 5.3, show that our algorithm solves both IPCE and 2-PCE for self-dual code instances similarly to the case of random code instances (with trivial hull).

4.4 Comparisons with Saeed's Algorithm [33]

In [33, Section 3.7], Saeed proposed an algorithm to solve PCE for random code instances. Let the following

$$(\mathbf{G}_1 = (\mathbf{I}_k | \mathbf{M}), \mathbf{G}_2 = \text{SF}(\mathbf{G}_1 \mathbf{P}) = (\mathbf{I}_k | \mathbf{M}')),$$

be a PCE instance, where $\mathbf{P} \in \text{Perm}_n(\mathbb{F}_q)$. From this only sample, they construct the following linear system

$$\begin{bmatrix} (\mathbf{I}_k | \mathbf{M}) \otimes (-\mathbf{M}'^\top | \mathbf{I}_{n-k}) \\ (-\mathbf{M}^\top | \mathbf{I}_{n-k}) \otimes (\mathbf{I}_k | \mathbf{M}') \\ \mathbf{I}_n \otimes \mathbf{1}_n^\top \\ \mathbf{1}_n^\top \otimes \mathbf{I}_n \end{bmatrix} \text{vec}(\mathbf{P}) = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1}_n \\ \mathbf{1}_n \end{bmatrix}, \quad (14)$$

where $\mathbf{1}_n$ is the column vector of length n and 1 in each entry. Notice that the first equations block is analogous to the one of Eq. (7). The second block is obtained thanks to the following observation: since $\mathbf{P}^{-1} = \mathbf{P}^\top$, we have that $\mathbf{G}_1 = \text{SF}(\mathbf{G}_2 \mathbf{P}^{-1}) = \text{SF}(\mathbf{G}_2 \mathbf{P}^\top)$ also holds. However, these new equations are, in general, linearly independent from the above only when the hull of the code is trivial. The last two equation blocks simply condition the sum of the elements of \mathbf{P} in the same row and column to equal 1, which is true for every permutation matrix.

Starting from the system in Eq. (14), Saeed’s algorithm works similarly to ours. However, our algorithm proposes a more efficient method for recovering the final secret (Line 14 of Algorithm 1). Specifically, our heuristic analysis shows that the number of survival variables is smaller than or equal to the number of equations, allowing an efficient recovery of the secret via Gaussian elimination. In contrast, the author of [33] does not present such an analysis, and they also do not specify how to recover the final solution. They, in fact, speculate that retrieving the solution may be computationally expensive as this step may require an exhaustive search on a large set.⁵

5 Experiments

We support the findings presented in this manuscript with extensive experiments and simulations performed by means of a SageMath [38] proof-of-concept implementation available at [13]. Regarding Sect. 3, we provide the scripts to test the correctness of Lemmas 1 and 2 and Corollaries 1 and 2. For Sect. 4, we report in this section the results of extensive experiments performed on solving 2-LCE/ILCE with random codes, and 2-PCE/IPCE with self-dual codes.

5.1 Solving 2-LCE

We perform extensive experiments to corroborate the weaker security provided by 2-LCE and ILCE when compared to LCE. We take into consideration the following observation on the parameter set from [2]:

- 128 bits: $n = 252$ and $q = 127$ satisfies $q \approx n/2$,
- 192 bits: $n = 400$ and $q = 127$ satisfies $q \approx n/3$,
- 256 bits: $n = 548$ and $q = 127$ satisfies $q \approx n/4$.

⁵ In [33, page 62], the author says “*This might have high complexity depending on the size of the solution set.*” We interpret this as requiring an exhaustive search.

To the best of our knowledge, the concrete security of 2-LCE was not analyzed before this work, and therefore we test our results on 2-LCE using the parameters providing different security levels for LCE. Thus, we focus on the following parameter set: $n \in [32, 40, 48, 64, 72, 80, 96, 128]$, $k = n/2$, and $q \in [n/2, n/3, n/4, 127]$. Essentially, we tackle cases that are believed to provide security equivalent to 20–70 bits in the case of LCE; such a complexity estimation is based on the analysis presented in [2]. Table 3 presents our experiments' time and memory measurements on a 2.45 GHz AMD EPYC 7763 64-core Processor machine with 1T of RAM running Ubuntu 22.04.2 LTS.

From some preliminary experiments, we observed that the upper bound in Proposition 1 is reached with overwhelming probability, i.e., $\text{rank}(\mathbf{A}_{ij}) = \frac{n^2}{2} - 1$. Hence, we optimize the algorithm and avoid one rank computation per guess by substituting the Rouché-Capelli test with the test of checking whether $\text{rank}(\mathbf{A}_{ij}|\mathbf{b}_{ij}) = \frac{n^2}{2} - 1$ or not.

Our implementation employs parallelization per row; more precisely, it runs n processors in parallel, and the j^{th} processor has the task of computing the rank of $\mathbf{S}_{i,j}$. Consequently, that parallelization approach gives a factor of n times faster, but the memory increases by the same factor (i.e., it is n times bigger). We use the multiprocessing Python package for the parallelization and the tracemalloc Python module to measure the memory usage. In addition, for each parameter set considered, Table 3 reports a comparison of the expected number of variables in \mathbf{S}_{red} against the average obtained in our experiments. This comparison illustrates that our experimental findings align with the analysis presented in Sect. 4.

5.2 Solving ILCE

We report the results of the experiments that we performed to support our claims in Sect. 4.2, i.e., Algorithm 1 solves ILCE analogously to 2-LCE. For different values of n and q , we report in Table 4 the measured success rate over 100 trials of both problems. One can see that 2-LCE and ILCE get solved with approximately the same success probability and that this corresponds to the success condition probability of Algorithm 1 (Eq. (9)).

5.3 Solving Self Dual 2-PCE and IPCE Instances

In this section, we report the results of our experiments to support our claim in Sect. 4.3, that is, Algorithm 1 solves 2-PCE and IPCE with self-dual codes instances analogously to random codes instances.

We consider the set of self-dual codes generators provided in [20,21], for $n \in \{16, 24, 28, 36, 40, 44\}$, $k = n/2$, and $q = 7$. Given that, for each n only one generator \mathbf{G} is given, we compute different 2-PCE instances at every test iteration as follows. First, we compute the generator of an equivalent code \mathbf{G}_1 of \mathbf{G} through a random permutation \mathbf{T} , and then we compute a PCE instance as $(\mathbf{G}_1, \mathbf{G}_2 = \text{SF}(\mathbf{G}_1\mathbf{P}))$, where \mathbf{P} is the random secret permutation to discover.

Table 3. The data corresponds to the average of solving 20 random 2-LCE instances. The fourth and the fifth columns present the expected number of variables in S_{red} according to Eq. (10) and the average of the observed values, respectively. The last column presents the number of successfully solved random 2-LCE instances (i.e., the success ratio obtained from the experiments).

n	q	Estimated LCE bit security	Expected vars in S_{red}	Measured vars in S_{red}	Memory (GB)	Runtime	Ratio
32	7	20	178	178	1.03	20s	18/20
	11	22	125	124	1.02	19s	14/20
	17	23	92	93	1.03	19s	19/20
	127	29	40	40	1.05	19s	20/20
40	11	25	185	183	2.57	48s	20/20
	13	25	163	165	2.56	47s	20/20
	19	27	124	121	2.56	47s	19/20
	127	33	53	54	2.57	47s	19/20
48	13	28	225	231	5.34	01m 41s	19/20
	17	29	183	173	5.35	01m 44s	18/20
	23	31	148	146	5.34	01m 44s	19/20
	127	37	66	69	5.36	01m 43s	20/20
64	17	35	305	288	16.96	07m 08s	17/20
	23	37	242	240	16.96	07m 00s	17/20
	31	38	196	191	16.96	07m 06s	20/20
	127	44	96	97	16.97	07m 02s	20/20
72	19	39	345	343	27.19	13m 27s	20/20
	23	40	297	291	27.19	13m 58s	17/20
	37	42	212	212	27.20	12m 50s	18/20
	127	47	113	113	27.21	13m 08s	20/20
80	19	41	416	417	41.48	21m 40s	18/20
	29	44	301	302	41.50	21m 48s	20/20
	41	46	236	228	41.49	18m 37s	18/20
	127	51	130	132	41.50	18m 09s	20/20
96	23	48	496	499	86.10	01h 04m	20/20
	31	51	393	392	86.10	01h 04m	19/20
	47	54	292	284	86.10	01h 04m	20/20
	127	58	169	169	86.09	01h 08m	20/20
128	31	63	656	639	272.06	06h 02m	20/20
	43	66	509	519	272.07	06h 02m	19/20
	61	69	397	397	272.06	05h 51m	19/20
	127	73	257	252	272.10	04h 39m	20/20

Table 4. The data corresponds to the number of solved instances divided by the total number of experiments (which is 100). The last column reports the expected success probability from our analysis, that is, the system in Eq. (7) is full-rank. In all the experiments, we have $k = n/2$.

q	n						$1 - \frac{1}{q}$
		16	24	32	40		
7	2-LCE	0.81	0.84	0.81	0.86	0.86	
	ILCE	0.87	0.82	0.86	0.85		
11	2-LCE	0.92	0.87	0.93	0.87	0.91	
	ILCE	0.91	0.93	0.89	0.90		
17	2-LCE	0.95	0.95	0.93	0.92	0.94	
	ILCE	0.96	0.94	0.96	0.96		
31	2-LCE	0.96	0.99	0.96	0.95	0.97	
	ILCE	0.94	0.96	0.98	0.98		

Table 5 reports the success rate over 100 trials, for the available values of n . One can note that our algorithm succeeds with probability approximately equal to $1 - \frac{1}{q} \approx 0.86$, matching the probability of our success condition, that is, the coefficient matrix in Eq. (13) is full-rank (see Eq. (9)).

Table 5. The data corresponds to the number of solved self-dual instances divided by the total number of experiments (which is 100). In all the experiments, we have $q = 7$ and $k = n/2$.

n	16	24	28	36	40	44
2-PCE	0.85	0.86	0.88	0.91	0.87	0.89
IPCE	0.85	0.83	0.88	0.84	0.90	0.86

6 Cryptographic Implications

To better illustrate the impact of the results from Sect. 4, we start by giving a comparison between the estimated complexities of LCE according to [2], and the complexity for 2-LCE and ILCE according to Sect. 4. We follow the parameter sets from [2], ensuring 128, 192, and 256 security bits for LCE under the current most efficient algorithms for solving it. On the other hand, the estimations from Sect. 4 imply a security of 2-LCE and ILCE of around 60–70 security bits for the same parameter sets (see Table 6).

On the Impact on ILCE-Based Linkable Signatures: In [4], the authors stated that if the ILCE problem were proved to be safe, all the necessary linkable properties

Table 6. The column corresponding to LCE is according to the security analysis from [2]. The column corresponding to 2-LCE & ILCE concerns the complexity of Algorithm 1 (detailed in Sect. 4) with $\omega = \log_2(7)$. The presented numbers are given in logarithm base two.

n	k	q	LCE	2-LCE & ILCE
252	126	127	128	61
400	200	127	192	66
548	274	127	256	70

would be satisfied, thus building a secure linkable ring signature scheme. Nevertheless, as a direct consequence of Sect. 4.2, we have that any linkable signature relying on the hardness of the ILCE problem is insecure when the conditions from Sect. 4 are satisfied.

On the Impact on 2-LCE-Based Threshold Signatures: The authors of [7] introduced the 2-LCE problem in the group action framework [7, Problem 3] and emphasized constructions for 2-weakly pseudorandom scenarios. Specifically, they proposed a threshold signature whose distributed key generation algorithm is based on the conjectured 2-weakly pseudorandom group actions built on top of the LCE and MCE problems. Nevertheless, as another consequence of Sect. 4, we show that Definition 7 when instantiated with group action based on LCE does not achieve the pseudorandomness property as we can use Algorithm 1 to recover the secret, which breaks the unpredictability as well as the pseudorandomness of the group action. Therefore, the threshold signature instantiations with LESS from [7, Sec. 5.3] become insecure when $k = n/2$.⁶

Other Implications: D’Alconzo and Di Scala have demonstrated that the LCE and MCE group actions do not guarantee weak unpredictability and weak pseudorandomness properties [19]. However, their findings do not apply when the instances are given in systematic form. Our work addresses this gap by providing a more general framework that includes the systematic form case. In light of this, Table 1 summarizes the primitives that, with instantiations from the literature, can and cannot be constructed using these group actions.

What About the Implications to 2-MCE? Given that the secret matrices of 2-MCE instances do not have the monomial structure, the algorithms from Sect. 4 do not apply to solving 2-MCE instances. In particular, it is unclear how to perform a similar guessing on the entries of the secret matrices.

Acknowledgments. Giuseppe D’Alconzo and Antonio J. Di Scala are members of GNSAGA of INdAM and of CryptTO, the group of Cryptography and Number Theory of the Politecnico di Torino.

⁶ The authors published an updated version of their protocol that does not rely on 2-LCE as a preprint after our attack was made public [6].

The work of Antonio J. Di Scala was partially supported by the QUBIP project (<https://www.qubip.eu>), funded by the European Union under the Horizon Europe framework programme [grant agreement no. 101119746].

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

We would also like to thank Andrea Natale and Ricardo Pontaza for their insights and discussions, which helped us improve the analysis of our techniques. Finally, we thank the anonymous reviewers of a previous version of this manuscript who provided us with helpful comments and recommendations.

References

1. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai and Wang [26], pp. 411–439. https://doi.org/10.1007/978-3-030-64834-3_14
2. Baldi, M., Beckwith, A.B.L., Biasse, J.F., Esser, A., Gaj, K., Mohajerani, K., Pelosi, G., Persichetti, E., Saarinen, M.J.O., Santini, P., Wallace, R.: LESS (version 1.1). Tech. rep., National Institute of Standards and Technology (2023), <https://www.less-project.com/>
3. Bardet, M., Otmani, A., Saeed-Taha, M.: Permutation Code Equivalence is Not Harder Than Graph Isomorphism When Hulls Are Trivial. In: 2019 IEEE International Symposium on Information Theory (ISIT). pp. 2464–2468 (2019). <https://doi.org/10.1109/ISIT.2019.8849855>
4. Barenghi, A., Biasse, J., Ngo, T., Persichetti, E., Santini, P.: Advanced signature functionalities from the code equivalence problem. *International Journal of Computer Mathematics: Computer Systems Theory* **7**(2), 112–128 (2022). <https://doi.org/10.1080/23799927.2022.2048206>
5. Barenghi, A., Biasse, J.F., Persichetti, E., Santini, P.: On the computational hardness of the code equivalence problem in cryptography. *Advances in Mathematics of Communications* **17**(1), 23–55 (2023). <https://doi.org/10.3934/amc.2022064>
6. Battagliola, M., Borin, G., Meneghetti, A., Persichetti, E.: Cutting the GRASS: Threshold GRoup Action Signature Schemes. *Cryptology ePrint Archive*, Paper 2023/859 (2023), <https://eprint.iacr.org/2023/859>
7. Battagliola, M., Borin, G., Meneghetti, A., Persichetti, E.: Cutting the grass: Threshold group action signature schemes. In: Oswald, E. (ed.) *Topics in Cryptology – CT-RSA 2024*. pp. 460–489. Springer Nature Switzerland, Cham (2024), https://doi.org/10.1007/978-3-031-58868-6_18
8. Benčina, B., Budroni, A., Chi-Domínguez, J.J., Kulkarni, M.: Properties of Lattice Isomorphism as a Cryptographic Group Action. In: *International Conference on Post-Quantum Cryptography*. pp. 170–201. Springer (2024), https://doi.org/10.1007/978-3-031-62743-9_6
9. Beullens, W.: Not enough LESS: An improved algorithm for solving code equivalence problems over \mathbb{F}_q . In: *International Conference on Selected Areas in Cryptography*. pp. 387–403. Springer (2020), https://doi.org/10.1007/978-3-030-81652-0_15
10. Beullens, W., Katsumata, S., Pintore, F.: Calamari and Falaff: Logarithmic (linkable) ring signatures from isogenies and lattices. In: Moriai and Wang [26], pp. 464–492. https://doi.org/10.1007/978-3-030-64834-3_16

11. Biasse, J.F., Micheli, G., Persichetti, E., Santini, P.: LESS is more: Code-based signatures without syndromes. In: Nitaj, A., Youssef, A.M. (eds.) AFRICACRYPT 20. LNCS, vol. 12174, pp. 45–65. Springer, Heidelberg (Jul 2020). https://doi.org/10.1007/978-3-030-51938-4_3
12. Bos, J.W., Bronchain, O., Ducas, L., Fehr, S., Huang, Y.H., Pornin, T., Postlethwaite, E.W., Prest, T., Pulles, L.N., van Woerden, W.: Hawk version 1.0 (june 1, 2023). Tech. rep., National Institute of Standards and Technology (2023), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/hawk-spec-web.pdf>
13. Budroni, A., Chi-Domínguez, J.J., D’Alconzo, G., Di Scala, A.J., Kulkarni, M.: `relaxed-lce-algorithms`, available at <https://github.com/JJChiDguez/relaxed-lce-algorithms.git>
14. Chavez-Saab, J., Santos, M.C.R., Feo, L.D., Eriksen, J.K., Hess, B., Kohel, D., Leroux, A., Longa, P., Meyer, M., Panny, L., Patranabis, S., Petit, C., Henríquez, F.R., Schaeffler, S., Wesolowski, B.: Squisign version 1.0 (june 1, 2023). Tech. rep., National Institute of Standards and Technology (2023), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/squisign-spec-web.pdf>
15. Chou, T., Niederhagen, R., Persichetti, E., Ran, L., Hajatiana, T., Reijnders, K., Samardjiska, S., Trimoska, M.: MEDS (version 1.1). Tech. rep., National Institute of Standards and Technology (2023), <https://www.meds-pqc.org/>
16. Chou, T., Niederhagen, R., Persichetti, E., Randrianarisoa, T.H., Reijnders, K., Samardjiska, S., Trimoska, M.: Take your MEDS: digital signatures from matrix code equivalence. In: Mrabet, N.E., Feo, L.D., Duquesne, S. (eds.) Progress in Cryptology - AFRICACRYPT 2023 - 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19-21, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14064, pp. 28–52. Springer (2023). https://doi.org/10.1007/978-3-031-37679-5_2
17. Chou, T., Persichetti, E., Santini, P.: On Linear Equivalence, Canonical Forms, and Digital Signatures. Cryptology ePrint Archive, Paper 2023/1533 (2023), <https://eprint.iacr.org/2023/1533>
18. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006), <https://eprint.iacr.org/2006/291>
19. D’Alconzo, G., Di Scala, A.J.: Representations of group actions and their applications in cryptography. *Finite Fields and Their Applications* **99**, 102476 (2024). <https://doi.org/10.1016/j.ffa.2024.102476>
20. Gaborit, P., Otmani, A.: TABLES OF SELF-DUAL CODES, available at https://www.unilim.fr/pages_perso/philippe.gaborit/SD/
21. Gaborit, P., Otmani, A.: Experimental constructions of self-dual codes. *Finite Fields and Their Applications* **9**(3), 372–394 (2003). [https://doi.org/10.1016/S1071-5797\(03\)00011-X](https://doi.org/10.1016/S1071-5797(03)00011-X)
22. Joux, A.: MPC in the head for isomorphisms and group actions. Cryptology ePrint Archive, Paper 2023/664 (2023), <https://eprint.iacr.org/2023/664>
23. Kazmi, R.A.: Cryptography from post-quantum assumptions. Cryptology ePrint Archive, Report 2015/376 (2015), <https://eprint.iacr.org/2015/376>
24. Leon, J.: Computing automorphism groups of error-correcting codes. *IEEE Transactions on Information Theory* **28**(3), 496–511 (1982). <https://doi.org/10.1109/TIT.1982.1056498>
25. Leroux, A., Roméas, M.: Updatable encryption from group actions. In: International Conference on Post-Quantum Cryptography. pp. 20–53. Springer (2024), https://doi.org/10.1007/978-3-031-62746-0_2

26. Moriai, S., Wang, H. (eds.): ASIACRYPT 2020, Part II, LNCS, vol. 12492. Springer, Heidelberg (Dec (2020))
27. National Institute of Standards and Technology: Post-Quantum Cryptography Standardization. <https://csrc.nist.gov/projects/post-quantum-cryptography> (2017)
28. National Institute of Standards and Technology: Post-quantum cryptography: Digital signature schemes. Round 1 Additional Signatures (2023), <https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures>
29. Persichetti, E., Randrianariso, T.H., Santini, P.: An attack on a non-interactive key exchange from code equivalence. *Tatra Mountains Mathematical Publications* **82**(2), 53–64 (2023). <https://doi.org/10.2478/tmmp-2022-0018>
30. Persichetti, E., Santini, P.: A New Formulation of the Linear Equivalence Problem and Shorter LESS Signatures. In: Guo, J., Steinfeld, R. (eds.) *Advances in Cryptology – ASIACRYPT 2023*. pp. 351–378. Springer Nature Singapore, Singapore (2023), https://doi.org/10.1007/978-981-99-8739-9_12
31. Petrank, E., Roth, R.M.: Is code equivalence easy to decide? *IEEE Transactions on Information Theory* **43**(5), 1602–1604 (1997). <https://doi.org/10.1109/18.623157>
32. Reijnders, K., Samardjiska, S., Trimoska, M.: Hardness Estimates of the Code Equivalence Problem in the Rank Metric. *Designs, Codes and Cryptography* pp. 1–30 (01 2024). <https://doi.org/10.1007/s10623-023-01338-x>
33. Saeed, M.A.: Algebraic Approach for Code Equivalence. Ph.D. thesis, Normandie Université, University of Khartoum, (2017), Available at <https://theses.hal.science/tel-01678829v2>
34. Santini, P., Baldi, M., Chiaraluce, F.: Computational hardness of the permuted kernel and subcode equivalence problems. *IEEE Transactions on Information Theory* **70**(3), 2254–2270 (2024). <https://doi.org/10.1109/TIT.2023.3323068>
35. Sendrier, N.: On the dimension of the hull. *SIAM Journal on Discrete Mathematics* **10**(2), 282–293 (1997). <https://doi.org/10.1137/S0895480195294027>
36. Sendrier, N.: Finding the permutation between equivalent linear codes: the support splitting algorithm. *IEEE Transactions on Information Theory* **46**(4), 1193–1203 (2000). <https://doi.org/10.1109/18.850662>
37. Sendrier, N., Simos, D.E.: The hardness of code equivalence over \mathbb{F}_q and its application to code-based cryptography. In: Gaborit, P. (ed.) *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*. pp. 203–216. Springer Heidelberg (June 2013), https://doi.org/10.1007/978-3-642-38616-9_14
38. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 9.8) (2023), <https://www.sagemath.org>



Rare Structures in Tensor Graphs

Bermuda Triangles for Cryptosystems Based on the Tensor Isomorphism Problem

Lars Ran^(✉) and Simona Samardjiska

Radboud University, Nijmegen, The Netherlands
{lran,simonas}@cs.ru.nl

Abstract. Recently, there has been a lot of interest in improving the understanding of the practical hardness of the 3-Tensor Isomorphism (3-TI) problem, which, given two 3-tensors, asks for an isometry between the two. The current state-of-the-art for solving this problem is the algebraic algorithm of Ran et al. '23 and the graph-theoretic algorithm of Narayanan et al. '24 that have both slightly reduced the security of the signature schemes MEDS and ALTEQ, based on variants of the 3-TI problem (Matrix Code Equivalence (MCE) and Alternating Trilinear Form Equivalence (ATFE) respectively).

In this paper, we propose a new combined technique for solving the 3-TI problem. Our algorithm, as typically done in graph-based algorithms, looks for an invariant in the graphs of the isomorphic tensors that can be used to recover the secret isometry. However, contrary to usual combinatorial approaches, our approach is purely algebraic. We model the invariant as a system of non-linear equations and solve it. Using this modelling we are able to find very rare invariant objects in the graphs of the tensors—cycles of length 3 (triangles)—that exist with probability approximately $1/q$. For solving the system of non-linear equations we use Gröbner-basis techniques adapted to tri-graded polynomial rings. We analyze the algorithm theoretically, and we provide lower and upper bounds on its complexity. We further provide experimental support for our complexity claims. Finally, we describe two dedicated versions of our algorithm tailored to the specifics of the MCE and the ATFE problems.

The implications of our algorithm are improved cryptanalysis of both MEDS and ALTEQ for the cases when a triangle exists, i.e. in approximately $1/q$ of the cases. While for MEDS, we only marginally reduce the security compared to previous work, for ALTEQ our results are much more significant with at least 60 bits improvement compared to previous work for all security levels. For Level I parameters, our attack is practical, and we are able to recover the secret key in only 1501 s. The code is available for testing and verification of our results.

Keywords: matrix codes · trilinear form · algebraic cryptanalysis

This research has been supported by the Dutch government through the NWO grant OCNW.M.21.193 (ALPaQCa).

© International Association for Cryptologic Research 2025
K.-M. Chung and Y. Sasaki (Eds.): ASIACRYPT 2024, LNCS 15491, pp. 66–96, 2025.
https://doi.org/10.1007/978-981-96-0944-4_3

1 Introduction

In recent years, all main standardization bodies around the world (NIST [31], ISO [25], IETF [24]) have initiated processes for standardization of post-quantum cryptography as the most solid solution for securing our digital world against the quantum computer menace. Post-quantum cryptography is the common name for cryptography based on hard mathematical problems believed to be hard even for quantum computers. NIST’s standardization process already produced drafts for standards—the winners Kyber [39], Dilithium [29], and SPHINCS+ [23] are well on the way to be standardized under the names of ML-KEM, ML-DSA, and SLH-DSA. The situation for Falcon [36] is not even close to this phase, and we have only recently seen some activity towards a draft standard. On top of this slow progress, NIST reopened the call for digital signatures in search for alternatives not based on structured lattices that additionally have short signatures and fast verification [1].

In the additional round that has been running for a year, we see an abundance of UOV variants, MPC-in-the-head Fiat-Shamir signatures and a few Fiat-Shamir signatures based on equivalence problems. In particular, MEDS [12, 13] and ALTEQ [8, 40] are based on problems equivalent to the 3-tensor isomorphism problem (TI) and LESS [3] on a problem at most as difficult as 3-TI [15]. Informally speaking, given two 3-tensors, the 3-TI problem asks for the isomorphism, i.e. isometry, between them. The shape of the isometry depends on the specific types of tensors in question. In the case of MEDS, these are general 3-tensors, and the isometry is given as a triple of the general linear group. The objects in MEDS can also be seen as matrix codes and the corresponding problem as Matrix Code Equivalence (MCE), which is actually the definition used in the original description of MEDS. In the case of ALTEQ, the objects can be seen as alternating trilinear forms (they can also be represented as matrix codes of skew-symmetric matrices with additional structure) and the isometry can be described using a single element of the general linear group. In this form, the problem is known as Alternating Trilinear Form Equivalence (ATFE).

Both of these schemes follow the well-known construction of GMW [20] first defined for graph isomorphism. Interestingly, already in ’96, it was instantiated by Patarin [33] for isomorphism of polynomials (Cubic-IP) whose version for homogeneous polynomials QMLE is also polynomial time equivalent to 3-TI, and thus also to MCE and ATFE [38]. Initially, the scheme did not get too much attention because of signature size inefficiency, but this changed due to an array of optimization techniques [3, 6, 7, 16] that were developed before the additional round submission deadline and resulted in the proposals of MEDS and ALTEQ. The question of practical hardness of these problems became interesting again, and the understanding of it significantly advanced as a result of these two submissions.

Related Work. The first works analyzing problems from the TI class, considered the Isomorphism of polynomials. As one of the problems intrinsically

related to the security of ad-hoc multivariate schemes, it was analyzed in several works including [10, 19, 34]. An important observation from [19] is that the inhomogeneous version of the problem can be solved heuristically in polynomial time. Another one is that knowing a single (point, image) of the isomorphism is enough to solve the homogeneous version of the problem. Indeed, this pair can be used to transform the instance of the problem to an inhomogeneous one that can then be solved efficiently. Bouillaguet et al. [10] gave a nice graph-theoretic interpretation of this observation as a matching point between the graphs of the two sets of polynomials which was utilized in a collision based algorithm.

The algorithm of [10] remained the best known for the TI class for a long time (note that the term ‘TI class’ was coined only recently in [21]). With the involvement of MEDS and ALTEQ in the NIST standardization process a significant advancement in the understanding of the asymptotic and practical hardness of the related underlying problems was made. We have witnessed basically the development of two types of algorithms—graph-based and purely algebraic. We say ‘purely algebraic’ because even the graph-based algorithms can employ an algebraic step for collecting low-rank points. This is especially beneficial for the case of large fields where the enumeration of points is expensive.

The graph-based algorithms build upon the earlier mentioned work of Bouillaguet et al. [10]. The first improvement was given by Beullens [5] for ATFE and by Chou et al. [13] for MCE. In both cases, the improvement follows the Leon’s algorithm for the Hamming metric [4, 28]. Currently, the state-of-the-art in graph-based algorithms is the work of Narayanan, Qiao and Tang [30], which presents two different algorithms for MCE and ATFE. The algorithm for ATFE is the one used for choosing the ALTEQ parameters from the specs. The one for MCE uses graph-walking techniques and notably, breaks the MEDS parameters submitted to NIST. As a result of this attack, very recently, at the NIST standardization conference [32], the MEDS team announced new parameters for all security levels.

The basic algebraic modeling for the MCE problem is somewhat of a folklore modeling known also from QMLE. The first non-trivial model was developed in [13] where coding theory relations were exploited. Later, in the MEDS specs [12] the modeling was improved by exploiting that 3-tensors give rise to 3 different matrix codes. The same approach was used in [37] but adapted to ATFE effectively reducing the security of the ALTEQ parameters. At the time of writing, the ALTEQ team has not announced new parameters as a reaction to the attack.

Our Contribution. In this work, we propose a new graph-based algebraic technique for solving Tensor Isomorphism (TI) problems. In particular, we algebraically find a rare invariant in the graph of the two isomorphic tensors and use it to find the isometry between the tensors. By an invariant, we mean a property or an object that is a characteristic of the graph of the tensor, and that is not destroyed by an isometric transformation, i.e. it is an *invariant* with respect to

isometries. Previous graph-theoretic algorithms use invariants like vertex degree or long paths to recover the isometry, and they typically consist of two parts:

1. Searching for occurrences of the invariant property within the two graphs of the two isomorphic tensors and forming two lists L_1 and L_2 containing corresponding points.
2. Testing each pair $(a, b) \in L_1 \times L_2$ whether b is an image of a for some isometry. If this is the case, we have found the isometry.

Typically, for this to work, the testing needs to be efficient, often, polynomial of low degree. Since the graphs of the tensors are large, enumerating the entire graphs is typically not feasible, so the algorithms rely on invariants that are abundant and easy to find. Using a birthday argument, it can then be estimated how big the lists need to be in order to have a high likelihood of finding a collision in the lists.

The first contribution of our work is that we take a different approach to finding invariants—namely, an algebraic one. We model the invariant as a system of non-linear equations—once we solve the system, we get the invariant. Using this modelling, we are able to find very rare, (almost) unique objects in the entire graph that would otherwise take an exponential amount of time to find by enumeration, which is particularly prohibitive in big fields. In contrast, an algebraic approach is oblivious to the field size up to the cost of the arithmetic which scales only logarithmically.

An important step in the approach is determining the right invariant object to look for. Indeed, we need one that is rare and can be described using a relatively small number of variables, but at the same time, we can impose enough restrictions in the form of algebraic relations. We show that with probability of approximately $1/q$, where q is the field size, there exist in the graphs of the tensors (almost) unique invariant objects. These objects are small cycles of length 3, i.e. triangles. This is reminiscent of Beullens' attack [5] that exploits the fact that with probability $\approx 1/q$, there exists a unique point of rank 4 in the graph of an alternating trilinear form in dimension $n = 10$.

Our second contribution is developing an algorithm for exploiting the existence of triangles. Our algorithm consists of two parts:

- First, we find the triangles in the graphs of the tensors. Once we have the algebraic model of the triangles, we solve the system using Gröbner basis techniques. The system we obtain has a specific structure and in order to solve it we develop a solving algorithm and a machinery for estimating the costs by extending known techniques for bigraded polynomial rings to trigraded polynomial rings. We identify the syzygies characteristic of the modeling and conjecture the Hilbert series. We are able to precisely estimate the first degree fall of the system which we take as an indicator of the solving costs or lower bound of our algorithm. As usual, we take the solving degree as an upper bound.
- Using the found pair of matching triangles we find the secret isometry. For the second part of the algorithm, we show that from the pair of triangles we can

obtain linear relations in the algebraic modelling of [12, 37] which are enough to heuristically find the isometry in polynomial time. Thus, this second part is significantly cheaper than the first part.

We have fully implemented the algorithm in MAGMA [9] which is publicly available at:

<https://github.com/LarsMath/tensor-triangles>

For the ATFE problem, our code can be used to demonstrate practicality. The git further contains the MAGMA source code for all experiments which we performed to verify our theoretical claims and confirm our conjectures. All the numbers in this paper can be reproduced using the code in the git.

Our third contribution is applying our new algorithm to the two digital signature schemes MEDS and ALTEQ whose security relies on problems equivalent to 3-TI. For the submitted parameter of MEDS we only marginally improve upon [30] for keys exhibiting a triangle. Recently, the MEDS team increased the parameters as a result of the attack from [30], which, due to the small difference, are also secure from our attack. The results are shown in Table 6 and Table 7.

The impact for ALTEQ is much more dramatic as can be seen from Table 1. We improve the best previous attack by at least ≈ 60 bits. Even more, we are able to practically break Level I parameters (provided a triangle exists, which happens with a probability of $\approx 1/q$). The attack takes merely 1501 s. The implementation of the full attack is also available in the git repository above.

Table 1. The \log_2 complexity for solving ATFE (with probability $1/q$) in field operations. The parameters are taken from the ALTEQ specifications [8]

		[8]	[37]		This work	
	n	Specs	Best previous	Upper bound	First degree fall	practical
Level I	13	143	120	62	51	1501 s
Level III	20	219	165	108	96	
Level V	25	276	203	141	128	

Organization. The paper is organized as follows. Sect. 2 introduces the necessary preliminaries including an analysis of the tri-graded XL that we will use in our analysis. In Sect. 3 we recall the state-of-the-art algorithms for solving the TI problem. Our new algorithm is developed in Sect. 4 in which we also directly estimate the impact on MCE and MEDS. We apply our approach to ATFE and ALTEQ in Sect. 5. We conclude with a discussion on potential generalizations and future work in Sect. 6.

2 Preliminaries

Let us first establish some notation. We denote by \mathbb{F}_q the finite field of q elements. By $\text{GL}(V)$ we denote the general linear group on the vector space V . The space of $n \times m$ matrices over \mathbb{F}_q are denoted by $\mathcal{M}_{n,m}(\mathbb{F}_q)$. We use bold letters to denote vectors $\mathbf{a}, \mathbf{c}, \mathbf{x}, \dots$, and capital bold letters to denote matrices $\mathbf{A}, \mathbf{B}, \dots$. The entries of a vector \mathbf{a} are denoted by a_i and the entries of a matrix \mathbf{A} are denoted by a_{ij} . We denote by $\mathbf{e}_1, \dots, \mathbf{e}_n$ the vectors of the canonical basis of \mathbb{F}_q^n . By $\mathbb{P}(V)$ we denote the projective space associated to a vector space V . To distinguish between vectors from V and elements from $\mathbb{P}(V)$, we denote the latter by $\hat{\mathbf{v}}$. All the vector spaces that we consider are finite-dimensional.

2.1 The Tensor Isomorphism Problem (TI) and Related Problems

Among cryptographic hardness assumptions based on equivalence, quite a few of them can be stated as a TI problem and especially as a 3-TI problem. For example, Matrix Code Equivalence (MCE), Alternating Trilinear Form Equivalence (ATFE), Quadratic Maps Linear Equivalence (QMLE) and Cubic Isomorphism of Polynomials (Cubic-IP) are all some form of 3-TI in disguise. In order to define 3-TI, we first need to define tensor isomorphisms.

Definition 1. *Given vector spaces U, V , and W over a field \mathbb{F}_q , a 3-tensor \mathcal{C} over U, V, W is a trilinear map:*

$$\mathcal{C} : U \times V \times W \rightarrow \mathbb{F}_q.$$

We denote the space of tensors over U, V, W as $(U \otimes V \otimes W)^$.*

Note that, by tri-linearity, a 3-tensor \mathcal{C} is completely determined by its values on the basis vectors of U, V, W , i.e. by $\mathcal{C}(\mathbf{e}_i^U, \mathbf{e}_j^V, \mathbf{e}_k^W)$ where $\mathbf{e}_i^U, 1 \leq i \leq \dim(U)$, $\mathbf{e}_j^V, 1 \leq j \leq \dim(V)$, and $\mathbf{e}_k^W, 1 \leq k \leq \dim(W)$ are basis vectors of U, V and W respectively. With this definition, we are ready to define the 3-TI problem.

Problem 1 (3-TI). Let U, V , and W be vector spaces over a field \mathbb{F}_q and let $\mathcal{C}, \mathcal{D} \in (U \otimes V \otimes W)^*$ be two given 3-tensors. The 3-TI problem asks to find, if any exists, a triplet of matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \text{GL}(U) \times \text{GL}(V) \times \text{GL}(W)$ such that:

$$\mathcal{C}(\mathbf{A}\mathbf{u}, \mathbf{B}\mathbf{v}, \mathbf{C}\mathbf{w}) = \mathcal{D}(\mathbf{u}, \mathbf{v}, \mathbf{w}) \quad \forall \mathbf{u} \in U, \mathbf{v} \in V, \mathbf{w} \in W.$$

Let us compare this to MCE.

Problem 2 (MCE). Let $n, m, k \geq 2$. Given matrices $C_1, \dots, C_k, D_1, \dots, D_k \in \mathcal{M}_{n,m}(\mathbb{F}_q)$ the MCE problem asks to find, if any exists, a pair of matrices $\mathbf{A}, \mathbf{B} \in \text{GL}(\mathbb{F}_q^n) \times \text{GL}(\mathbb{F}_q^m)$ such that:

$$\langle \mathbf{A}^\top C_1 \mathbf{B}, \dots, \mathbf{A}^\top C_k \mathbf{B} \rangle = \langle D_1, \dots, D_k \rangle.$$

Note the similarity between 3-TI and MCE. In fact, these problems are equivalent [22] and \mathbf{C} in 3-TI is exactly the change of basis from $\langle \mathbf{A}^\top C_1 \mathbf{B}, \dots, \mathbf{A}^\top C_k \mathbf{B} \rangle$ to $\langle D_1, \dots, D_k \rangle$. Other variants of 3-TI can be obtained by limiting the space of tensors $(U \otimes V \otimes W)^*$. For example, taking $U = V = W$, we can impose that the tensors from $(V \otimes V \otimes V)^*$ must be alternating, i.e. we ask:

$$\mathcal{C}(\mathbf{v}, \mathbf{v}, \mathbf{w}) = \mathcal{C}(\mathbf{v}, \mathbf{w}, \mathbf{v}) = \mathcal{C}(\mathbf{w}, \mathbf{v}, \mathbf{v}) = 0 \text{ for all } \mathbf{v}, \mathbf{w} \in V.$$

We denote the space of tensors that satisfy this constraint by $(\bigwedge^3 V)^*$. Now we can state the ATFE problem similarly as 3-TI:

Problem 3 (ATFE). Let V be a vector space over a field \mathbb{F}_q and let $\phi, \psi \in (\bigwedge^3 V)^*$ be two given alternating 3-tensors. The ATFE problem asks to find, if it exists, a matrix $\mathbf{A} \in \text{GL}(V)$ such that:

$$\phi(\mathbf{A}\mathbf{u}, \mathbf{A}\mathbf{v}, \mathbf{A}\mathbf{w}) = \psi(\mathbf{u}, \mathbf{v}, \mathbf{w}) \quad \forall \mathbf{u}, \mathbf{v}, \mathbf{w} \in V.$$

An immediate consequence of the alternating property is that compared to 3-TI we now need $\mathbf{A} = \mathbf{B} = \mathbf{C}$. The other two problems, QMLE and Cubic-IP, come from considering partly symmetric and fully symmetric 3-tensors.

Remark 1. The two signature schemes MEDS and ALTEQ do not assume any further structure on the codes (alternating forms) and isometries they use. Hence, we too, will be interested in unstructured (random) variants of these problems.

2.2 Graphs Associated with Tensors

A strong invariant of a 3-tensor is the graph associated with it. The points of this graph are given by elements in the disjoint union of the projective vector spaces. Its edges consist of the pairs of points on which the tensor completely vanishes. To define this we will use the following shorthand notation. Given a tensor $\mathcal{C} : U \times V \times W \rightarrow \mathbb{F}_q$, the statement $\mathcal{C}(\mathbf{u}, \mathbf{v}, -) = 0$ denotes that $\mathcal{C}(\mathbf{u}, \mathbf{v}, \mathbf{x}) = 0$ for all $\mathbf{x} \in W$. We define $\mathcal{C}(\mathbf{u}, -, \mathbf{w}) = 0$ and $\mathcal{C}(-, \mathbf{v}, \mathbf{w}) = 0$ similarly. Note that the statement $\mathcal{C}(\mathbf{u}, \mathbf{v}, -) = 0$ is independent of scaling of \mathbf{u} and \mathbf{v} , so we will use the notation $\mathcal{C}(\hat{\mathbf{u}}, \hat{\mathbf{v}}, -) = 0$ as well.

Definition 2. Let $\mathcal{C} : U \times V \times W \rightarrow \mathbb{F}_q$ be a tensor. The graph $\mathcal{G}(\mathcal{C}) = (\mathcal{V}_\mathcal{C}, \mathcal{E}_\mathcal{C})$ is defined as follows:

$$\begin{aligned} \mathcal{V}_\mathcal{C} &= \mathbb{P}(U) \uplus \mathbb{P}(V) \uplus \mathbb{P}(W) \\ \mathcal{E}_\mathcal{C} &= \{(\hat{\mathbf{u}}, \hat{\mathbf{v}}) \in \mathbb{P}(U) \times \mathbb{P}(V) \mid \mathcal{C}(\hat{\mathbf{u}}, \hat{\mathbf{v}}, -) = 0\} \\ &\quad \cup \{(\hat{\mathbf{u}}, \hat{\mathbf{w}}) \in \mathbb{P}(U) \times \mathbb{P}(W) \mid \mathcal{C}(\hat{\mathbf{u}}, -, \hat{\mathbf{w}}) = 0\} \\ &\quad \cup \{(\hat{\mathbf{v}}, \hat{\mathbf{w}}) \in \mathbb{P}(V) \times \mathbb{P}(W) \mid \mathcal{C}(-, \hat{\mathbf{v}}, \hat{\mathbf{w}}) = 0\} \end{aligned}$$

From the definition, we can immediately see that these graphs are tripartite with vertex partition $(\mathbb{P}(U), \mathbb{P}(V), \mathbb{P}(W))$. A particularly useful feature about these associated graphs is that this construction is functorial. In other words, suppose

we have two 3-tensors $\mathcal{C} : U \times V \times W \rightarrow \mathbb{F}_q$ and $\mathcal{D} : U' \times V' \times W' \rightarrow \mathbb{F}_q$ and a transformation $\mathbf{A} : U' \rightarrow U$, $\mathbf{B} : V' \rightarrow V$, $\mathbf{C} : W' \rightarrow W$ between them, such that:

$$\mathcal{D} = \mathcal{C} \circ (\mathbf{A}, \mathbf{B}, \mathbf{C})$$

Then we obtain a map on graphs by applying the matrices \mathbf{A} , \mathbf{B} , and \mathbf{C} on the vertices. It is then easy to check that this map indeed maps edges to edges. The map $(\mathbf{A}, \mathbf{B}, \mathbf{C}) : \mathcal{G}(\mathcal{D}) \rightarrow \mathcal{G}(\mathcal{C})$ is given by:

$$(\mathbf{A}, \mathbf{B}, \mathbf{C}) : v \mapsto \begin{cases} \mathbf{A}v & \text{if } v \in \mathbb{P}(V') \\ \mathbf{B}v & \text{if } v \in \mathbb{P}(W') \\ \mathbf{C}v & \text{if } v \in \mathbb{P}(U'). \end{cases}$$

The important takeaway here is that isomorphisms of 3-tensors, elements of $\text{GL}(U) \times \text{GL}(V) \times \text{GL}(W)$, induce isomorphisms on the associated graphs.

Graphs Associated with Structured Tensors. When there is a structure present in the tensor, for example, (anti-)symmetry, then the above construction has a lot of superfluous points and edges. For example, consider the alternating trilinear form $\phi : V \times V \times V \rightarrow \mathbb{F}_q$, and two elements $\mathbf{v}, \mathbf{v}' \in \mathbb{P}(V)$ such that $\phi(\mathbf{v}, \mathbf{v}', -) = 0$. Then by anti-symmetry we also have $\phi(\mathbf{v}, -, \mathbf{v}') = 0$, and similarly for any other permutation of \mathbf{v}, \mathbf{v}' , and $-$. In other words, without labels, it is impossible to distinguish whether \mathbf{v} lies in the first, second or third term of $\mathbb{P}(V) \uplus \mathbb{P}(V) \uplus \mathbb{P}(V)$. More precisely, there are 6 graph automorphisms permuting the terms in the disjoint union. Therefore, instead, we consider the vertex set of the graph to be $\mathcal{V}_\phi = \mathbb{P}(V)$. For other types of symmetries, we consider similar quotient graphs.

2.3 Gröbner Basis Algorithms

To obtain solutions to the polynomial systems we encounter below, we will use Gröbner basis algorithms [2, 11, 14, 18, 26, 27]. These algorithms come in all kinds of variants, but they all share the same underlying idea. The goal is to gather enough algebraic combinations of the initial polynomials such that linear combinations of these reduce the problem to a linear system. Generally, it is hard to say how many algebraic combinations one needs to do this. However, heuristically, we can make some approximations based on assumptions about (structured) random systems.

Macaulay Matrices. Let us consider a polynomial system $\mathcal{F} = (f_1, \dots, f_m) \subset \mathbb{F}_q[x_1, \dots, x_n] = \mathcal{R}$. We are interested in the spaces of algebraic combinations of \mathcal{F} up to a certain degree:

$$I_{\leq d} := \text{span}_{\mathbb{F}_q} \{u \cdot f \mid u \in \mathcal{R}, f \in \mathcal{F}, \deg(uf) \leq d\}.$$

If $I_{\leq d}$ contains n linear independent linear elements, we reduced our problem to a linear system. If it contains the element $1_{\mathcal{R}}$ then we know that our system does not have a solution.

Algorithmically, to find the solution, we build the Macaulay matrix \mathcal{M}_d of degree d . This matrix has its columns labeled by the monomials up to degree d in \mathcal{R} . The rows are given by the products uf_i where $\deg(u) \leq d - \deg(f)$. The entry at (uf_i, m) is then the coefficient of the monomial m in the product uf_i .

Note that this is generally a sparse matrix, as multiplying with a monomial does not change the amount of terms. Also note that the rowspace of \mathcal{M}_d is isomorphic to $I_{\leq d}$. Therefore, to check that $I_{\leq d}$ contains linear relations we can echolonize \mathcal{M}_d and see if we end up with rows having only linear monomials.

The question now is how high d needs to be for this to happen. To estimate this we need to assume that we can exactly predict the amount of linear dependencies, called syzygies, among the rows of \mathcal{M}_d . Even though this assumption might look strong, in practice, for random systems, the rank of \mathcal{M}_d neatly follows a pattern for different n, m, d . Then, we will obtain a solution exactly when we predict \mathcal{M}_d to be of corank 1. We will call the degree in which this happens the solving degree d_{solv} . To then extract the linear system we need to echolonize this matrix which has $\binom{n+d_{\text{solv}}}{d_{\text{solv}}}$ columns. Since this matrix is sparse we can use the Block-Wiedemann algorithm to obtain a complexity of:

$$\rho \cdot \binom{n + d_{\text{solv}}}{d_{\text{solv}}}^2$$

where ρ is the density of the matrix and is equal to the maximum number of terms among the polynomials in \mathcal{F} .

Degree Falls. Instead of choosing d such that we can completely linearize, some algorithms, like F4 and MutantXL, take a different strategy. In these cases we are looking for *degree falls* which happen when the vector space

$$I_d := \text{span}_{\mathbb{F}_q} \{u \cdot f \mid u \in \mathcal{R}, f \in \mathcal{F}, \deg(uf) = d\}$$

has elements of degree smaller than d . The degree at which this happens is called the first fall degree d_{ff} . Note that this can only happen in inhomogeneous systems. Then, we can extend the Macaulay matrix in degree d_{ff} by adding these elements multiplied by linear monomials. In this way, the system might be solved in lower degree.

The complexity analysis of these algorithms and how they behave after finding degree falls is much more complex. Still, it is not uncommon to take the complexity of finding degree falls, $\rho \cdot \binom{n+d_{ff}-1}{d_{ff}}^2$, as an estimator for the complexity of solving the system.

Tri-graded XL. Due to the structure of the polynomial systems that we will consider, a tri-graded polynomial ring is often a better fit. These are rings

$$\mathcal{R} = \mathbb{F}_q[x_1, \dots, x_n, y_1, \dots, y_m, z_1, \dots, z_k]$$

where the grading is determined by the degree in each of the three variables sets $\{x_1, \dots, x_n\}, \{y_1, \dots, y_m\}, \{z_1, \dots, z_k\}$. In this ring, we index monomials by $\alpha \in \mathbb{Z}_{\geq 0}^n, \beta \in \mathbb{Z}_{\geq 0}^m, \gamma \in \mathbb{Z}_{\geq 0}^k$ and we use the following notation:

$$\mathbf{x}^\alpha \mathbf{y}^\beta \mathbf{z}^\gamma = \prod_{i=1}^n x_i^{\alpha_i} \prod_{i=1}^m y_i^{\beta_i} \prod_{i=1}^k z_i^{\gamma_i}$$

for a monomial of tri-degree $(\sum_i \alpha_i, \sum_i \beta_i, \sum_i \gamma_i)$.

A polynomial is tri-homogeneous of tri-degree (d_x, d_y, d_z) if all its terms share the same tri-degree. As an example, a tri-homogeneous polynomial in degree $(1, 1, 1)$ is exactly a trilinear form! We sometimes drop the tri prefix and use homogeneous and degree if it is clear from the context what is meant.

Given two tri-degrees $\mathbf{d} = (d_x, d_y, d_z)$ and $\mathbf{d}' = (d'_x, d'_y, d'_z)$, we define the partial order $\mathbf{d} \succeq \mathbf{d}'$ if $(d_x \geq d'_x) \wedge (d_y \geq d'_y) \wedge (d_z \geq d'_z)$. If, among the degrees of the monomials of a polynomial f , there is a greatest tri-degree, then we call this degree the top degree of f .

Just as for singly graded systems, we can define

$$I_{\preceq \mathbf{d}} := \text{span}_{\mathbb{F}_q} \{u \cdot f \mid u \in \mathcal{R}, f \in \mathcal{F}, \text{deg}(uf) \preceq \mathbf{d}\}$$

and corresponding Macaulay matrices. Then, instead of a single lowest degree for which this is linearizable, we may have multiple “lowest” tri-degrees. Let us call a tri-degree *admissible* if $I_{\preceq \mathbf{d}}$ contains $n + m + k$ linear relations or the element 1. Let us denote by $\mathcal{D}_{\text{solv}}$ the set of all admissible tri-degrees for \mathcal{F} . Then the complexity of linearizing can be given by:

$$\rho \cdot \min_{\mathbf{d} \in \mathcal{D}_{\text{solv}}} \left(\binom{n + d_x}{d_x} \binom{m + d_y}{d_y} \binom{k + d_z}{d_z} \right)^2.$$

Just as before we can define the first fall degree. However, since, again, there might be multiple tri-degrees for which this is the case, we write the complexity as:

$$\rho \cdot \min_{\mathbf{d} \in \mathcal{D}_{\text{ff}}} \left(\binom{n + d_x - 1}{d_x} \binom{m + d_y - 1}{d_y} \binom{k + d_z - 1}{d_z} \right)^2.$$

Remark 2. Estimating the impact of degree falls in tri-graded systems is even more complex than in singly-graded systems. The resulting polynomials do not have a unique top degree anymore hence usual counting strategies fail. However, it is still clear that degree falls can only speed up computation.

3 Algorithms for Solving TI

The algorithms against TI build upon relatively old algorithms against the Isomorphism of polynomials [10, 35]. Here we review the state-of-the-art.

3.1 Graph-Theoretic Algorithm of Narayanan Et Al. [30]

The work of Narayanan, Qiao and Tang [30] builds on top of the works of [5, 10] and presents two different algorithms for MCE and ATFE. On a high level, both algorithms follow the structure of the graph-theoretic algorithm of [10] and can be described as follows:

- Form the graphs of the two isomorphic tensors \mathcal{C} and \mathcal{D} (matrix codes or alternating trilinear forms) as described in Sect. 2.2.
- Collect points from the two graphs into two lists $L_{\mathcal{C}}$ and $L_{\mathcal{D}}$ such that the bilinear forms obtained by fixing the tensor at the given point is of specific rank R . Due to the birthday paradox, the size of the lists needs to be only a square root of all points satisfying the rank R property.
- For each element in $L_{\mathcal{C}}$ and $L_{\mathcal{D}}$ apply some sort of distinguishing function f constructed in such a way that $f(a) = f(b)$ only if (a, b) is a collision pair for the unknown isometry
- Use the collision pair (a, b) for which $f(a) = f(b)$ to recover the isometry

The difference from [10] which is also the main novelty of this approach, is the formulation of the distinguishing function. Previously this function was just solving an inhomogeneous QMLE for every possible pair in $L_{\mathcal{C}} \times L_{\mathcal{D}}$. With this approach, there is no need to test each pair, but one can make full use of the birthday paradox and just look for a collision in the lists.

Besides this novel global invariant (as called in [30]), there are also some new interesting techniques introduced. For example in the algorithm for MCE, in order to construct the distinguishing function, the authors use a graph walking technique to efficiently find a path of length $3n$. Until this work, it was an open question of how to use graph walking techniques to solve MCE. Previously, graph walking has been efficiently used against ATFE by Beullens in [5].

In some sense, looking for distinguishing cycles in the two graphs was an inspiration for our work. However, we take one more shortcut, and look for a unique cycle in the two graphs.

3.2 Purely Algebraic Algorithms for Solving TI

The described graph-theoretic algorithms in the previous subsection crucially rely on algebraic techniques. For example, the inhomogeneous efficient solver is purely algebraic, and most likely, can't be replaced by an equally efficient combinatorial procedure. Also, the enumeration part of low-rank codewords can be done much more efficiently by solving the MinRank problem algebraically.

There are, however, also purely algebraic approaches developed in [12, 13] for MCE and [8, 37, 40] for the related ATFE. Here the solution is modelled as part of a solution to a system of equations.

Taking the coefficients of the matrices $\mathbf{A}, \mathbf{A}', \mathbf{B}, \mathbf{B}'$, and \mathbf{C}, \mathbf{C}' as unknowns, we can build the following system of equations:

$$\begin{cases} \mathbf{A}\mathbf{A}' = \mathbf{I}_n = \mathbf{A}'\mathbf{A} \\ \mathbf{B}\mathbf{B}' = \mathbf{I}_m = \mathbf{B}'\mathbf{B} \\ \mathbf{C}\mathbf{C}' = \mathbf{I}_k = \mathbf{C}'\mathbf{C} \end{cases} \quad (1)$$

$$\begin{cases} \mathcal{C}(\mathbf{A}\mathbf{x}, \mathbf{B}\mathbf{y}, \mathbf{z}) = \mathcal{D}(\mathbf{x}, \mathbf{y}, \mathbf{C}'\mathbf{z}) \\ \mathcal{C}(\mathbf{A}\mathbf{x}, \mathbf{y}, \mathbf{C}\mathbf{z}) = \mathcal{D}(\mathbf{x}, \mathbf{B}'\mathbf{y}, \mathbf{z}) \\ \mathcal{C}(\mathbf{x}, \mathbf{B}\mathbf{y}, \mathbf{C}\mathbf{z}) = \mathcal{D}(\mathbf{A}'\mathbf{x}, \mathbf{y}, \mathbf{z}) \end{cases} \quad \forall \mathbf{x} \in \mathbb{F}_q^n, \mathbf{y} \in \mathbb{F}_q^m, \mathbf{z} \in \mathbb{F}_q^k \quad (2)$$

$$\begin{cases} \mathcal{C}(\mathbf{A}\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathcal{D}(\mathbf{x}, \mathbf{B}'\mathbf{y}, \mathbf{C}'\mathbf{z}) \\ \mathcal{C}(\mathbf{x}, \mathbf{B}\mathbf{y}, \mathbf{z}) = \mathcal{D}(\mathbf{A}'\mathbf{x}, \mathbf{y}, \mathbf{C}'\mathbf{z}) \\ \mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{C}\mathbf{z}) = \mathcal{D}(\mathbf{A}'\mathbf{x}, \mathbf{B}'\mathbf{y}, \mathbf{z}) \end{cases} \quad \forall \mathbf{x} \in \mathbb{F}_q^n, \mathbf{y} \in \mathbb{F}_q^m, \mathbf{z} \in \mathbb{F}_q^k \quad (3)$$

This is an immense quadratic system of $6nmk + 2(n^2 + m^2 + k^2)$ equations in $2(n^2 + m^2 + k^2)$ variables. However, there is a lot of superfluity in this system. By construction, the syzygy module in degree 3 is quite big.

Solving the system “as is” has a high complexity, the number one reason being that it has a huge amount of variables. So instead, the current best purely algebraic algorithm from [12] looks at a subsystem generated as follows.

Consider the equations in (2). Here the left-hand side is quadratic in $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and the right-hand side is linear in $\mathbf{A}', \mathbf{B}', \mathbf{C}'$. This means we can take linear combinations to eliminate $\mathbf{A}', \mathbf{B}', \mathbf{C}'$. Then we end up with a system of $3nmk - (n^2 + m^2 + k^2)$ equations, quadratic in $n^2 + m^2 + k^2$ variables, $\mathbf{A}, \mathbf{B}, \mathbf{C}$. These equations are tri-homogeneous and hence the following Hilbert series is conjectured:

$$\mathcal{H}(r, s, t) = \frac{(1 - rs)^{nmk - k^2} (1 - rt)^{nmk - m^2} (1 - st)^{nmk - n^2} (1 - rst)^{-\alpha}}{(1 - r)^{n^2} (1 - s)^{m^2} (1 - t)^{k^2}}.$$

Here $\alpha = 2nmk - n^2 - m^2 - k^2 + 1$ is the dimension of the syzygy module in tri-degree $(1, 1, 1)$.

Specializing to ATFE. For ATFE a similar system is obtained in the unknown \mathbf{A} and \mathbf{A}' :

$$\begin{cases} \mathbf{A}\mathbf{A}' = \mathbf{I}_n = \mathbf{A}'\mathbf{A} \\ \phi(\mathbf{A}\mathbf{x}, \mathbf{A}\mathbf{y}, \mathbf{z}) = \psi(\mathbf{x}, \mathbf{y}, \mathbf{A}'\mathbf{z}) \\ \phi(\mathbf{A}\mathbf{x}, \mathbf{y}, \mathbf{z}) = \psi(\mathbf{x}, \mathbf{A}'\mathbf{y}, \mathbf{A}'\mathbf{z}) \end{cases} \quad \forall \mathbf{x} \in \mathbb{F}_q^n$$

This is a system of $2n\binom{n}{2} + 2n^2$ quadratic equations in $2n^2$ variables. However, the same trick of eliminating \mathbf{A}' can be applied to obtain a system of $n\left(\binom{n}{2} - n\right)$ quadratic equations in n^2 variables. The following Hilbert series is conjectured for this system:

$$\mathcal{H}(t) = \frac{(1 - t^2)^n \binom{n}{2}^{-n} (1 - t^3)^{-\beta}}{(1 - t)^{n^2}}.$$

Here $\beta = n \binom{n}{2} - \binom{n}{3} + 1$ is the dimension of the syzygy module in degree 3. This Hilbert series was experimentally verified for small n and d in [37] and seems to hold for n not too small.

4 A Hybrid Algorithm for Solving TI

As described in Sect. 2 and seen in Sect. 3 a strong invariant of a tensor is its graph. Most notably, any isometry between two tensors maps substructures in one graph to substructures in the second that have some common property. These can be, for example, points of a certain degree or an edge between two points of certain degrees. Identifying the substructures that are mapped one onto another (we will call them “collisions”) can be turned into an algorithm for finding the isometry.

However, two costly factors arise in such algorithms. The first is finding all (or a fraction of) such substructures in the graphs and the second is testing for collisions in some enumerative way. This generally involves a balancing act. Often, substructures of which there are few are hard to find and substructures that are easy to find are far from unique. The latter case becomes particularly prohibitive for big fields, since usually, the amount of substructures is highly dependent on the field size. So even though finding is easy, enumerating all candidates becomes the bottleneck. In contrast, when a substructure appears only once in a graph, finding it in both graphs immediately leads to a collision.

Our attack follows this general approach, but instead of exploiting abundant substructures, we strive to find such that are rare or unique in a graph. While we were not able to find a substructure that occurs exactly once, we found a substructure that occurs exactly once with sufficiently large probability of $\approx 1/q$ —we call this substructure a triangle. Then, if a triangle appears in the graph of a specific tensor, by invariance, it occurs in every graph of tensors in its orbit. Also, given that a triangle consists of three points, whenever we find such triangles, we get a three-point collision. As we will see, this three-point collision is enough to solve the corresponding TI problem efficiently.

4.1 Triangles

The rare structure that we consider is a triplet of points that form a triangle in the graph associated to the tensor. To define it formally, denote the set of all triplets of points in the graph by $\mathbb{T}(U, V, W) = \mathbb{P}(U) \times \mathbb{P}(V) \times \mathbb{P}(W)$. We have:

Definition 3. *Let $\mathcal{C} : U \times V \times W \rightarrow \mathbb{F}_q$ be a 3-tensor. We call a triplet of points $(\hat{\mathbf{u}}, \hat{\mathbf{v}}, \hat{\mathbf{w}}) \in \mathbb{T}(U, V, W)$ a triangle for \mathcal{C} if it holds that:*

$$\mathcal{C}(\hat{\mathbf{u}}, \hat{\mathbf{v}}, -) = 0, \quad \mathcal{C}(\hat{\mathbf{u}}, -, \hat{\mathbf{w}}) = 0, \quad \mathcal{C}(-, \hat{\mathbf{v}}, \hat{\mathbf{w}}) = 0.$$

Initially, it might seem that such a simple structure should be abundant in the graphs. However, it turns out that this is not the case at all.

Lemma 1. *Let U , V and W be vector spaces over \mathbb{F}_q . Then the expectation value for the amount of triangles is equal to:*

$$\mathbb{E}_{\mathcal{C} \in (U \otimes V \otimes W)^*} (|\{T \in \mathbb{T}(U, V, W) \mid T \text{ is a triangle for } \mathcal{C}\}|) = q^{-1}.$$

Proof. We follow a proof technique similar to [5]. We first find the probability that an arbitrary triplet $(\hat{\mathbf{u}}, \hat{\mathbf{v}}, \hat{\mathbf{w}}) \in \mathbb{T}(U, V, W)$ is a triangle, then, we use linearity of the expectation value to get the desired result. Given a point $T = (\hat{\mathbf{u}}, \hat{\mathbf{v}}, \hat{\mathbf{w}})$ we pick a non-zero representative $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ and extend it to a base. We obtain $U = \langle \mathbf{u}, \mathbf{u}_2, \dots, \mathbf{u}_n \rangle$, $V = \langle \mathbf{v}, \mathbf{v}_2, \dots, \mathbf{v}_m \rangle$, and $W = \langle \mathbf{w}, \mathbf{w}_2, \dots, \mathbf{w}_k \rangle$.

When we consider a tensor \mathcal{C} with respect to this basis we obtain a tensor \mathcal{C}' . If we pick \mathcal{C} uniformly at random then \mathcal{C}' will be uniformly random as well. Then, the condition that T is a triangle for \mathcal{C} is equivalent to the condition that $(\mathbf{e}_1^U, \mathbf{e}_1^V, \mathbf{e}_1^W)$ is a triangle for \mathcal{C}' . This latter condition can be reformulated as

$$\mathcal{C}(\mathbf{e}_{i_1}^U, \mathbf{e}_1^V, \mathbf{e}_1^W) = \mathcal{C}(\mathbf{e}_1^U, \mathbf{e}_{i_2}^V, \mathbf{e}_1^W) = \mathcal{C}(\mathbf{e}_1^U, \mathbf{e}_1^V, \mathbf{e}_{i_3}^W) = 0$$

for all $1 \leq i_1 \leq n, 1 \leq i_2 \leq m, 1 \leq i_3 \leq k$. See Fig. 1 for a depiction. Since these values are all independently uniform in \mathbb{F}_q we obtain that:

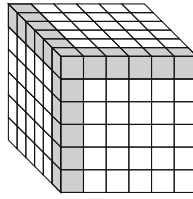


Fig. 1. For a 3-tensor \mathcal{C} , the coefficients in the light gray positions should be zero in order for $(\mathbf{e}_1^U, \mathbf{e}_1^V, \mathbf{e}_1^W)$ to be a triangle.

$$\mathbb{P}_{\mathcal{C} \in (U \otimes V \otimes W)^*} (T \text{ is a triangle for } \mathcal{C}) = q^{-(n+m+k-2)}.$$

Now by simple linearity of the expectation value we obtain:

$$\begin{aligned} & \mathbb{E}_{\mathcal{C} \in (U \otimes V \otimes W)^*} (|\{T \in \mathbb{T}(U, V, W) \mid T \text{ is a triangle for } \mathcal{C}\}|) \\ &= \mathbb{P}_{\mathcal{C} \in (U \otimes V \otimes W)^*, T \in \mathbb{T}(U, V, W)} (T \text{ is a triangle}) \cdot |\mathbb{T}(U, V, W)| \\ &= q^{-(n+m+k-2)} \cdot q^{n-1} \cdot q^{m-1} \cdot q^{k-1} \\ &= q^{-1}. \end{aligned}$$

Of course, the expectation value does not directly give us the probability that a triangle occurs, let alone a unique one. Therefore, we also show a lower bound for this probability.

Corollary 1. *Given vector spaces U , V , and W over \mathbb{F}_q of dimensions n , m , and k such that $n \geq m \geq k \geq 3$ and $m + k - 2 \geq n$. Then:*

$$\mathbb{P}_{\mathcal{C} \in (U \otimes V \otimes W)^*}(\mathcal{C} \text{ has a unique triangle}) \geq q^{-1} - \mathcal{O}(q^{-2}).$$

Proof. See Appendix A.1.

To support this result we did a Monte-Carlo simulation of the amount of tensors with (unique) triangles using MAGMA's `VarietySize`, the results can be found in Table 2. For practical reasons, triangles of which the first \mathbf{u} , \mathbf{v} , or \mathbf{w} coordinate is 0 were not found, so actual numbers might be higher.

Table 2. Fraction of tensors where a triangle was found on 10000 experiments. We also report the times the triangle was unique.

q	(n, m, k)	Predicted	Found	Unique
13	(5,5,5)	769	739	713
	(6,5,5)	769	755	725
	(6,6,6)	769	692	676
31	(5,5,5)	323	301	196
	(6,5,5)	323	306	303
	(6,6,6)	323	298	290
251	(5,5,5)	40	46	46
	(6,5,5)	40	33	33
	(6,6,6)	40	43	43

Remark 3. The constraint $m + k - 2 \geq n$ is not limiting. The probability that a tensors graph has any (V, W) edge, which is necessary for a triangle, is upper bounded by $q^{(m-1)+(k-1)-n}$. So if instead $m + k - 1 \leq n$, this probability is upper bounded by q^{-1} . If such an edge would exist it would serve as a better invariant of the tensor than a triangle. Even more, it would be easier to find.

4.2 Finding Triangles

A major reason why previous algorithms do not exploit rare or unique graph structures is the difficulty of finding them. Searching the whole graph is prohibitively expensive and birthday-based techniques can not be used since the structures are extremely rare.

To overcome this difficulty, we propose to look for these algebraically. For the triangles we defined above, the problem can be modeled as a system of quadratic equations in $R = \mathbb{F}_q[x_2, \dots, x_n, y_2, \dots, y_m, z_2, \dots, z_k]$. Let us denote

$\mathbf{x} = [1, x_2, \dots, x_n]$, $\mathbf{y} = [1, y_2, \dots, y_m]$ and $\mathbf{z} = [1, z_2, \dots, z_k]$. Indeed, we can consider the following system:

$$\begin{cases} \mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{e}_i^W) = 0 & \text{for } 1 \leq i \leq k \\ \mathcal{C}(\mathbf{x}, \mathbf{e}_i^V, \mathbf{z}) = 0 & \text{for } 1 \leq i \leq m \\ \mathcal{C}(\mathbf{e}_i^U, \mathbf{y}, \mathbf{z}) = 0 & \text{for } 1 \leq i \leq n. \end{cases}$$

Then, if we find a solution to the system, the point $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is a representative of the triangle. The attentive reader might find that we are not able to find all triangles in this way. Whenever the first coordinate of \mathbf{u} , \mathbf{v} , or \mathbf{w} is zero. However, we simply ignore this issue, since one can rerandomize by applying a random transformation to the given tensor.

Looking at our system we see that we found $n + m + k$ quadratic equations in $n + m + k - 3$ variables. We can now go and use our favorite system-solving technique to find all such solutions. The technique that we will use to compute the complexity is tri-homogeneous XL, see Sect. 2.3.

We consider the system in the following three sets of variables $\{x_2, \dots, x_n\}$, $\{y_2, \dots, y_m\}$, and $\{z_2, \dots, z_k\}$. Then the system described above consists of n equations in each of the tri-degrees $(1, 1, 0)$, $(1, 0, 1)$, and $(0, 1, 1)$. Furthermore, if we set $x_1 = y_1 = z_1 = 1$ we have the following 2 syzygies in tri-degree $(1, 1, 1)$:

$$\sum_{i=1}^k \mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{e}_i^U) \cdot z_i = \sum_{i=1}^m \mathcal{C}(\mathbf{x}, \mathbf{e}_i^W, \mathbf{z}) \cdot y_i = \sum_{i=1}^n \mathcal{C}(\mathbf{e}_i^V, \mathbf{y}, \mathbf{z}) \cdot x_i.$$

Therefore, we conjecture the following Hilbert series:

$$\mathcal{H}(r, s, t) = \frac{(1 - rs)^n (1 - rt)^n (1 - st)^n (1 - rst)^{-2}}{(1 - r)^{n-1} (1 - s)^{m-1} (1 - t)^{k-1}}.$$

We confirmed the predicted number of syzygies in all tri-degrees (d_x, d_y, d_z) for several n, m, k . For simplicity, we only checked parameters that are in use in cryptography, so $n = m = k$ and $n = m + 1 = k + 1$. For the $(8, 7, 7)$ case we find some wrong predictions. This could be due to extra structure for small n . The results are summarized in Table 3. In Table 4, one can find the results of running MAGMA's `GroebnerBasis` on the described system for some small values of n, m , and k . As we can see the solving degree neatly matches the predicted first fall degree, indicating that this is a valid estimator for the complexity. Note that `GroebnerBasis` does not take the specialized tri-graded structure into account. Hence, specialized implementations, like some form of Tri-XL, could speed up computation even more.

4.3 From Matching Triangles to Isometry

Now let us see what we can do with such a triangle. Recall our problem statement, we need to find the isometry $(\mathbf{A}, \mathbf{B}, \mathbf{C}) : \mathcal{C} \rightarrow \mathcal{D}$. As stated before, if \mathcal{C} (and thus \mathcal{D}) has no triangle, then this attack is impossible. If the triangles do

Table 3. These indicate the experimental nullities in each tri-degree. All values that were predicted wrong are formatted (predicted/actual).

Parameters	Tri-degree						
	(1, 1, 1)	(2, 1, 1)	(3, 1, 1)	(2, 2, 1)	(4, 1, 1)	(3, 2, 1)	(2, 2, 2)
(7,7,7)	2	61	336	772	1141	—	—
(8,8,8)	2	78	504	1174	1960	6356	11601
(9,9,9)	2	97	720	1694	3156	10512	DNF
(8,7,7)	2	63	399	882	1540	4599/4620	7969/8064
(9,8,8)	2	80	584	1316	2544	7896	13907

Table 4. Finding triangles in 3-TI.

(n, m, k)	Actual			Predicted	
	Time	Memory	d_{solv}	\mathbf{d}_{solv}	\mathbf{d}_{ff}
(6,6,6)	4 s	32 MB	5	(4,3,1)	(3,1,1)
(7,7,7)	163 s	288 MB	6	(6,3,1)	(3,2,1)
(8,8,8)	7 h	10 GB	7	(6,4,1)	(3,3,1)
(7,6,6)	7 s	64 MB	5	(4,3,1)	(3,1,1)
(8,7,7)	450 s	626 MB	6	(6,3,1)	(3,2,1)

exist but are not unique, then we iterate over the combinations. So let us assume that \mathcal{C} and \mathcal{D} have unique triangles $T_{\mathcal{C}}$ and $T_{\mathcal{D}}$. By applying suitable basis transformations we can transform these triangles to lie in any position. Therefore, without loss of generality we assume $T_{\mathcal{C}} = (\mathbf{e}_1^U, \mathbf{e}_1^V, \mathbf{e}_1^W) = T_{\mathcal{D}}$. Afterwards we just compose the solution with the inverse of the chosen basis transformations.

Since the isometry maps $T_{\mathcal{C}}$ to $T_{\mathcal{D}}$, we have that \mathbf{A}, \mathbf{B} , and \mathbf{C} have the following form:

$$\mathbf{A} = \begin{bmatrix} \lambda & \mathbf{A}_{12} \\ \mathbf{0}_{(n-1) \times 1} & \mathbf{A}_{22} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \mu & \mathbf{B}_{12} \\ \mathbf{0}_{(m-1) \times 1} & \mathbf{B}_{22} \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} \nu & \mathbf{C}_{12} \\ \mathbf{0}_{(k-1) \times 1} & \mathbf{C}_{22} \end{bmatrix} \quad (4)$$

For the inverses of the matrices, we have the form

$$\mathbf{A}' = \lambda^{-1} \begin{bmatrix} 1 & \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \\ \mathbf{0} & \lambda \mathbf{A}_{22}^{-1} \end{bmatrix} \quad \mathbf{B}' = \mu^{-1} \begin{bmatrix} 1 & \mathbf{B}_{12} \mathbf{B}_{22}^{-1} \\ \mathbf{0} & \mu \mathbf{B}_{22}^{-1} \end{bmatrix} \quad \mathbf{C}' = \nu^{-1} \begin{bmatrix} 1 & \mathbf{C}_{12} \mathbf{C}_{22}^{-1} \\ \mathbf{0} & \nu \mathbf{C}_{22}^{-1} \end{bmatrix}. \quad (5)$$

Here λ, μ, ν are some scalars in \mathbb{F}_q that come from the fact that our triangles live in projective space. However, we can freely rescale \mathbf{B} and \mathbf{C} by scaling \mathbf{A} accordingly, so without loss of generality we assume $\mu = \nu = 1$. Under these assumptions, we get that the triangle vectors are eigenvectors of the isometry matrices, more concretely, $\mathbf{A}\mathbf{e}_1 = \lambda\mathbf{e}_1$, $\mathbf{B}\mathbf{e}_1 = \mathbf{e}_1$, $\mathbf{C}\mathbf{e}_1 = \mathbf{e}_1$. Plugging these relations into the equations in (2) and (3), part of them become linear, i.e. we

obtain:

$$\begin{cases} \mathcal{C}(\mathbf{A}\mathbf{x}, \mathbf{e}_1^V, \mathbf{z}) = \mathcal{D}(\mathbf{x}, \mathbf{e}_1^V, \mathbf{C}'\mathbf{z}) \\ \mathcal{C}(\mathbf{x}, \mathbf{e}_1^V, \mathbf{C}\mathbf{z}) = \mathcal{D}(\mathbf{A}'\mathbf{x}, \mathbf{e}_1^V, \mathbf{z}) \end{cases} \quad \forall \mathbf{x} \in \mathbb{F}_q^n, \mathbf{z} \in \mathbb{F}_q^k \quad (6)$$

$$\begin{cases} \mathcal{C}(\mathbf{A}\mathbf{x}, \mathbf{y}, \mathbf{e}_1^W) = \mathcal{D}(\mathbf{x}, \mathbf{B}'\mathbf{y}, \mathbf{e}_1^W) \\ \mathcal{C}(\mathbf{x}, \mathbf{B}\mathbf{y}, \mathbf{e}_1^W) = \mathcal{D}(\mathbf{A}'\mathbf{x}, \mathbf{y}, \mathbf{e}_1^W) \end{cases} \quad \forall \mathbf{x} \in \mathbb{F}_q^n, \mathbf{y} \in \mathbb{F}_q^m \quad (7)$$

Notice that these correspond exactly to the top and front slices of the tensor. Furthermore, we also get the following, almost linear, equations (recall that λ is unknown):

$$\begin{cases} \mathcal{C}(\lambda\mathbf{e}_1^U, \mathbf{B}\mathbf{y}, \mathbf{z}) = \mathcal{D}(\mathbf{e}_1^U, \mathbf{y}, \mathbf{C}'\mathbf{z}) \\ \mathcal{C}(\lambda\mathbf{e}_1^U, \mathbf{y}, \mathbf{C}\mathbf{z}) = \mathcal{D}(\mathbf{e}_1^U, \mathbf{B}'\mathbf{y}, \mathbf{z}) \end{cases} \quad \forall \mathbf{y} \in \mathbb{F}_q^m, \mathbf{z} \in \mathbb{F}_q^k \quad (8)$$

Since the matrices $\mathcal{C}(\mathbf{x}, \mathbf{e}_1^V, \mathbf{z})$ are not full rank by construction, we know that not all the linear relations in Eq. (6) are independent. Similarly, for the Eq. (7) and Eq. (8). As we can see, a big chunk of the variables can already be eliminated by linear equations. On top of that, there is a substantial amount of quadratic equations that the solution has to satisfy. Instead of analyzing whether there are any algebraic dependencies between those linear and quadratic equations, we went with a more practical approach and ran a Gröbner basis algorithm on the resulting system. For all experiments, the system terminated in degree 2 so we conjecture that this is the solving degree for such systems. This would lead to a complexity of $\mathcal{O}(\binom{n^2+1}{2}^\omega) = \mathcal{O}(n^{4\omega})$. Here ω is the linear algebra constant. In any case, as can be seen from the experimental results in Table 5, this part of the algorithm is practical for all values of n for which we can hope to find a triangle.

Table 5. Running times for the post-collision algorithm. We also report the linear independent equations in the system presented in Eqs. (6), (7) and (8).

(n, m, k)	Variables	Linear equations	Quadratic equations	Time	d_{solv}
(8,8,8)	238	196	2802	2 s	2
(14,14,14)	1094	674	15962	245 s	2
(20,20,20)	2282	1444	47042	3 h	2
(9,8,8)	370	224	3156	68 s	2
(16,15,15)	1322	840	13566	536 s	2
(21,20,20)	2362	1520	49364	3 h	2

4.4 Putting It All Together

As we saw above, the cost of finding the triangle is the most dominating. If we take into account the search for an attack then we get the following complexity for the original MEDS parameters (Table 6):

Table 6. The \log_2 complexity estimates for the original MEDS parameters. All complexities are in field operations. We use the solving degree as an estimator here.

			[12]	[30]	This work	
	n	q	Specs	Best previous	incl. search	prob. $1/q$
Level I	14	4093	147	95	102	90
Level III	22	4093	217	145	155	143
Level V	30	2039	276	180	208	197

Recently, as a result of the attack from [30], the MEDS team updated the parameters [32]. In Table 7, we show the complexity results for the new parameters. As we can see, the new parameters are secure against this attack, even up to the first fall degree.

Table 7. Complexity estimations \log_2 for newest MEDS parameter set. As [30] was only shown to work for balanced parameters we estimate its complexity for the nearest balanced parameters.

				First fall degree		Solving degree		[30]	
	n	m	k	q	\mathbf{d}_{ff}	compl.	\mathbf{d}_{solv}	compl.	compl.
Level I	26	25	25	4093	(11, 18, 1)	151	(13, 20, 1)	165	164*
Level III	35	34	34	4093	(16, 25, 1)	210	(18, 27, 1)	224	219*
Level V	45	44	44	4093	(23, 31, 1)	275	(25, 33, 1)	289	280*

5 Application to ATFE

Having seen the above algorithm, a natural question is to ask whether it can also be applied to more structured tensors, like alternating tensors. It turns out, that indeed it does, and that after taking care of some technicalities, we can even use the extra structure in our advantage.

First, let us try the same as before, but now in the reduced graph. We look at triplets of points $(\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \hat{\mathbf{v}}_3) \in \mathbb{T}(V, V, V)$ for which

$$\phi(\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, -) = \phi(\hat{\mathbf{v}}_1, -, \hat{\mathbf{v}}_3) = \phi(-, \hat{\mathbf{v}}_2, \hat{\mathbf{v}}_3) = 0.$$

Now by anti-symmetry and tri-linearity this means that $\phi(\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2 + \hat{\mathbf{v}}_3, -) = 0$ and also $\phi(\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_1, -) = 0$. In other words, let $\hat{\mathbf{w}}_1, \hat{\mathbf{w}}_2, \hat{\mathbf{w}}_3$ be any linear combinations of $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \hat{\mathbf{v}}_3$, then $(\hat{\mathbf{w}}_1, \hat{\mathbf{w}}_2, \hat{\mathbf{w}}_3)$ is a triangle as well. Even more, $(\hat{\mathbf{v}}, \hat{\mathbf{v}}, \hat{\mathbf{v}})$ is a triangle for any $\hat{\mathbf{v}} \in \mathbb{P}(V)$.

This structure points us to look at a modified definition of a “triangle”. Let $\mathbb{T}(V) = \{W \subset V \mid \dim(W) = 3\}$ be the set of all 3-dimensional subspaces of V .

Definition 4. Let $\phi : \wedge^3 V \rightarrow \mathbb{F}_q$ be an alternating trilinear form. We call a 3-dimensional subspace $T \in \mathbb{T}(V)$ a triangle for ϕ if

$$\phi(\mathbf{v}, \mathbf{w}, -) = 0 \quad \forall \mathbf{v}, \mathbf{w} \in T.$$

This definition has been considered before in [17] in the pursuit of classifying alternating trilinear forms. There they were called 3-dimensional 2-singular subspaces. Just as in the unstructured case, we have the following:

Lemma 2. The expectation value for the number of such constructions in a random alternating trilinear form is equal to q^{-1} , i.e.:

$$\mathbb{E}_{\phi \in (\wedge^3 V)^*} (|\{T \in \mathbb{T}(V) \mid T \text{ is a triangle for } \phi\}|) = q^{-1}.$$

Proof. We follow a similar structure as in Lemma 1. We extend $\langle \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \rangle = T \in \mathbb{T}(V)$ to a basis $V = \langle \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \dots, \mathbf{v}_n \rangle$. Applying a basis transformation keeps the coefficients of ϕ uniformly random. So we need to compute the probability that $(\mathbf{e}_1^V, \mathbf{e}_2^V, \mathbf{e}_3^V)$ is a triangle for a random ϕ . This latter condition can be reformulated as $\phi_{123} = \phi_{12i} = \phi_{13i} = \phi_{23i} = 0$ for all $4 \leq i \leq n$. (See Fig. 2).

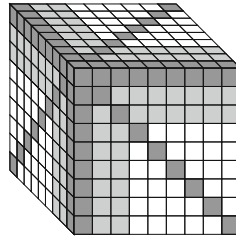


Fig. 2. For an ATF ϕ , the coefficients in the light gray positions should be zero for $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ to be a triangle. The coefficients corresponding to the dark gray positions are zero by alternatingness.

Taking symmetries into account these are exactly $3(n - 3) + 1$ relations. Since the coefficients are all uniform in \mathbb{F}_q and using simple linearity of the expectation value, we obtain:

$$\begin{aligned} &\mathbb{E}_{\phi \in (\wedge^3 V)^*} (|\{T \in \mathbb{T}(V) \mid T \text{ is a triangle for } \phi\}|) \\ &= \mathbb{P}_{\phi \in (\wedge^3 V)^*, T \in \mathbb{T}(V)} (T \text{ is a triangle for } \phi) \cdot |\mathbb{T}(V)| \\ &= q^{-(3(n-3)+1)} \cdot q^{3(n-3)} \\ &= q^{-1}. \end{aligned}$$

To say something about how reliably our new algorithm can be used, we need a stronger result, i.e., not only the expectation, but the probability of having a unique triangle. Fortunately, we have the following corollary:

Corollary 2. *Given a vector space V over \mathbb{F}_q of dimension $n = \dim(V) \geq 9$:*

$$\mathbb{P}_{\phi \in (\wedge^3 V)^*}(\phi \text{ has a unique triangle}) \geq q^{-1} - \mathcal{O}(q^{-2}).$$

Proof. See Appendix A.2.

We verify this experimentally using the polynomial system described below. The results, which are in line with our corollary, are given in Table 8. For practical reasons, we assume that the triangle does not intersect with the $x_1 = x_2 = x_3 = 0$ hyperplane. This allows us to fix variables to make solving computable. Again, MAGMA’s `VarietySize` is used on the system described in the next section.

Table 8. Fraction of tensors where a triangle was found on 10000 experiments. We also report the times the triangle was unique.

q	n	Predicted	Found	Unique
13	9	769	720	665
	10	769	716	694
	11	769	759	740
31	9	323	266	259
	10	323	311	308
	11	323	319	314
251	9	40	33	33
	10	40	33	33
	11	40	40	40

5.1 Finding Triangles

To find these triangles we are going to use algebraic methods again. In this case we will be working over the ring $\mathcal{R} = \mathbb{F}_q[x_4, \dots, x_n, y_4, \dots, y_n, z_4, \dots, z_n]$. We use a shorthand notation for the following vectors in \mathcal{R}^n :

$$\mathbf{x} = [1, 0, 0, x_4, \dots, x_n], \quad \mathbf{y} = [0, 1, 0, y_4, \dots, y_n], \quad \mathbf{z} = [0, 0, 1, z_4, \dots, z_n].$$

Now, given an alternating trilinear form ϕ , our system for finding a triangle looks as follows:

$$\begin{cases} f_{\mathbf{xy}}^{(i)} := \phi(\mathbf{x}, \mathbf{y}, \mathbf{e}_i) = 0 \\ f_{\mathbf{yz}}^{(i)} := \phi(\mathbf{y}, \mathbf{z}, \mathbf{e}_i) = 0 \\ f_{\mathbf{zx}}^{(i)} := \phi(\mathbf{z}, \mathbf{x}, \mathbf{e}_i) = 0 \end{cases} \quad \forall 1 \leq i \leq n \tag{9}$$

It is clear that when we have found a solution $(x_4, \dots, x_n, y_4, \dots, y_n, z_4, \dots, z_n)$ we indeed have also found a triangle.

We consider solving this system using Gröbner basis methods. To make the analysis precise we are going to conjecture a Hilbert series, but first, we find a class of structural syzygies. In tri-degree $(1, 1, 1)$, just as in the MCE case, we have the following 2 linear independent syzygies appearing:

$$\sum_{i=1}^n \phi(\mathbf{x}, \mathbf{y}, e_i) \cdot z_i = \sum_{i=1}^n \phi(\mathbf{y}, \mathbf{z}, e_i) \cdot x_i = \sum_{i=1}^n \phi(\mathbf{z}, \mathbf{x}, e_i) \cdot y_i.$$

However, in the ATFE case, there are 6 more syzygies, in tri-degrees that are a permutation of $(2, 1, 0)$. They are due to the alternating properties:

$$\sum_{i=1}^n \phi(\mathbf{x}, \mathbf{y}, e_i) \cdot x_i = \phi(\mathbf{x}, \mathbf{y}, \mathbf{x}) = 0$$

Taking everything together we conjecture the Hilbert series. For reasons that become apparent in Sect. 5.3 we consider the tri-homogenized version instead:

$$\begin{aligned} \mathcal{S}(r, s, t) &= (1 - r^2 s)(1 - r s^2)(1 - s^2 t)(1 - s t^2)(1 - t^2 r)(1 - t r^2)(1 - r s t)^2 \\ \mathcal{H}(r, s, t) &= \frac{(1 - r s)^n (1 - r t)^n (1 - s t)^n}{(1 - r)^{n-2} (1 - s)^{n-2} (1 - t)^{n-2}} \cdot \mathcal{S}^{-1}. \end{aligned}$$

We created the Macaulay matrix for different values of n, d_x, d_y , and d_z and computed their rank. With this, we verified the amount of linear independent syzygies predicted by the Hilbert series. The results are summarized in Table 9.

Table 9. Experimental syzygies in each tri-degree ($q = 29$). All values that were predicted wrong are formatted (predicted/actual).

n	Tri-degree								
	(2, 2, 0)	(2, 1, 1)	(4, 1, 0)	(3, 2, 0)	(3, 1, 1)	(2, 2, 1)	(5, 1, 0)	(4, 2, 0)	(3, 3, 0)
12	86	184	55	803	1726	4002	220/221	4281/4282	7241/7242
13	100	213	66	1032	2207	5140	286/287	6018/6019	10319/10320
14	115	244	78	1300	2768	6472	364	8231	14277
15	131	277	91	1610	3415	8013	455	10999	
16	148	312	105	1965	4154	9778	560		
17	166	349	120	2368	4991				
18	185	388	136	2822	5932				
19	205	429	153	3330					
20	226	472	171	3895					

Now, let us see what this means in practice. In Table 10, we report the running time of MAGMA’s GroebnerBasis on this system. As we can see, both \mathbf{d}_{solv} and \mathbf{d}_{ff} largely overestimate the actual solving degree. Given that the

experimental results on the Hilbert series are so affirmative, this might seem odd. What happens in practice is that there are quite some structural degree falls. We will see more on these in Sect. 5.3. For now, the main takeaway is that for $n = 13$ and $n = 14$, finding triangles is practical.

Table 10. Finding triangles in ATFE, experimental results ($q = 29$).

n	Actual			Predicted	
	Time	Memory	d_{solv}	\mathbf{d}_{solv}	\mathbf{d}_{ff}
12	29 s	96 MB	4	(7, 3, 1)	(5, 2, 1)
13	490 s	850 MB	4	(7, 4, 1)	(5, 3, 1)
14	30 h	29 GB	5	(7, 5, 1)	(5, 4, 1)
15	≥ 14 d	≥ 260 GB	≥ 6	(7, 6, 1)	(5, 5, 1)

5.2 Post-collision

Now, following the same line of argumentation as in Sect. 4.3, assume that ϕ and ψ both have a unique triangle in general position $\langle \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \rangle$. This places the following limit on any transformation $\mathbf{A} : \phi \rightarrow \psi$ between them:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{0}_{(n-3) \times 3} & \mathbf{A}_{22} \end{pmatrix} \quad \mathbf{A}^{-1} = \begin{pmatrix} \mathbf{A}_{11}^{-1} & -\mathbf{A}_{11}^{-1} \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \\ \mathbf{0}_{(n-3) \times 3} & \mathbf{A}_{22}^{-1} \end{pmatrix}$$

These restrictions unfortunately do not generate linear equations. However, instead, we consider this system as a tri-graded system in variable sets

$$\{\mathbf{A}_{11}, \mathbf{A}'_{11}\}, \quad \{\mathbf{A}_{12}, \mathbf{A}'_{12}\}, \quad \{\mathbf{A}_{22}, \mathbf{A}'_{22}\}.$$

Inspired by an unverified Hilbert series (not shown here) we use degree weights to put more emphasis on the $\{\mathbf{A}_{11}, \mathbf{A}'_{11}\}$ variables. Now running `GroebnerBasis` yields the desired results, these can be found in Table 11 below. Given the efficiency of solving $n = 25$, we do not optimize beyond this.

Table 11. Practical runtimes of the post-collision algorithm, $q = 2^{32} - 5$

n	Time (s)	Memory (MB)	$(wt_{11}, wt_{12}, wt_{22})$	weighted d_{solv}
13	102	864	(1, 3, 3)	6
20	2317	8287	(1, 3, 3)	6
25	8736	21750	(1, 3, 3)	6

Given that the complexity of the post-collision is negligible we provide the complexities for the complete algorithm in Table 12. These hold for the $1/q$ amount of ATFE problems that have a triangle.

Table 12. The \log_2 complexity for solving ATFE (with probability $1/q$) in field operations. The parameters are taken from the ALTEQ specifications [8].

		[8]	[37]	This work		
	n	Specs	Best previous	\mathbf{d}_{solv} compl.	\mathbf{d}_{ff} compl.	Time
Level I	13	143	120	62	51	1501 s
Level III	20	219	165	108	96	
Level V	25	276	203	141	128	

5.3 A Peculiar Set of Degree Falls

The results in Table 10 indicate that there is something more going on than our Hilbert series predicts. This has to do with the fact that there are additional degree falls present. In F4-like algorithms, these degree falls immediately speed up computation. In this section, we show some structural degree falls that happen for all values of n . First, we need to extend the definition of alternating multilinear forms to multivariate polynomials:

Definition 5. Let $f \in \mathbb{F}_q[\mathbf{x}, \mathbf{y}, \mathbf{z}]$, where $\mathbf{x} = (x_1, \dots, x_n)$, $\mathbf{y} = (y_1, \dots, y_n)$, $\mathbf{z} = (z_1, \dots, z_n)$. We call f alternating if

$$f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = -f(\mathbf{y}, \mathbf{x}, \mathbf{z}) = -f(\mathbf{z}, \mathbf{y}, \mathbf{x}).$$

Remark 4. Just as in the case of alternating trilinear forms, one can show that, in finite fields, this is equivalent to putting constraints on the coefficients of such polynomials. The polynomial $f = \sum_{\alpha, \beta, \gamma \in \mathbb{Z}_{\geq 0}^n} c_{\alpha\beta\gamma} \mathbf{x}^\alpha \mathbf{y}^\beta \mathbf{z}^\gamma$ is alternating if and only if

$$c_{\alpha\beta\gamma} = -c_{\beta\alpha\gamma} = -c_{\gamma\beta\alpha} \quad \forall \alpha, \beta, \gamma \in \mathbb{Z}_{\geq 0}^n.$$

As an example, the alternating tri-homogeneous polynomials living in tri-degree $(1, 1, 1)$ are exactly the alternating trilinear forms. A simple counting argument tells us that the dimension of tri-homogeneous alternating elements in tri-degree (d, d, d) is exactly $\binom{M_{n,d}}{3}$. Here $M_{n,d} = \binom{n+d-1}{d}$ is the amount of degree d monomials in n variables. For $d = 1$ we then indeed get $\binom{n}{3}$. Furthermore, we find that there are no tri-homogeneous alternating elements in tri-degree (d_x, d_y, d_z) if $d_x \neq d_y$ or $d_x \neq d_z$.

Now back to the degree falls. A degree fall happens when there is a syzygy among the homogeneous top parts of elements which is not a full syzygy. Recall our system Eq. (9). We can write the top-degree part of our system using the following substitutes

$$\bar{\mathbf{x}} = [0, 0, 0, x_4, \dots, x_n], \quad \bar{\mathbf{y}} = [0, 0, 0, y_4, \dots, y_n], \quad \bar{\mathbf{z}} = [0, 0, 0, z_4, \dots, z_n].$$

Then we can easily write down the top-degree parts of our system:

$$\overline{f_{\mathbf{xy}}^{(i)}} = f_{\mathbf{xy}}^{(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$$

To construct syzygies among these, we will build alternating functions using symmetry. We will use the following lemma.

Lemma 3. *Let $f \in \mathbb{F}_q[x_1, \dots, x_n, y_1, \dots, y_n, z_1, \dots, z_n]$ such that f is alternating in its first two arguments, i.e. $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = -f(\mathbf{y}, \mathbf{x}, \mathbf{z})$. Then,*

$$\mathcal{S}(f)(\mathbf{x}, \mathbf{y}, \mathbf{z}) := f(\mathbf{x}, \mathbf{y}, \mathbf{z}) + f(\mathbf{z}, \mathbf{x}, \mathbf{y}) + f(\mathbf{y}, \mathbf{z}, \mathbf{x})$$

is alternating.

In our system, the polynomials $f_{\mathbf{xy}}^{(i)}$ are indeed alternating in the first two arguments and hence the polynomials $\overline{f_{\mathbf{xy}}^{(i)}}$ are too. Furthermore, for the top-degree parts we have $\overline{f_{\mathbf{xy}}^{(i)}}(\mathbf{y}, \mathbf{z}, \mathbf{x}) = \overline{f_{\mathbf{yz}}^{(i)}}(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Therefore, the polynomial function

$$\mathcal{S}(\overline{f_{\mathbf{xy}}^{(i)}}) = \overline{f_{\mathbf{xy}}^{(i)}} + \overline{f_{\mathbf{yz}}^{(i)}} + \overline{f_{\mathbf{zx}}^{(i)}}$$

is alternating and a linear combination of the top degrees of polynomials in our system. We can take this a step further and consider $\mathcal{S}(\overline{f_{\mathbf{xy}}^{(i)}} \cdot g(\mathbf{z}))$ for a linear polynomial g . The result lies in degree $(1, 1, 1)$ and is, in fact, an alternating trilinear form. More generally, we can look at the subspace of the Macaulay space in $(1, 1, 1)$ generated by $\mathcal{S}(\overline{f_{\mathbf{xy}}^{(i)}} \cdot \mathbf{z}_j)$ for $1 \leq i \leq n$ and $1 \leq j \leq n - 3$. We know that this space is contained in $(\bigwedge^3 \mathbb{F}_q^{n-3})^*$. Hence, by the rank nullity theorem, we can conclude that there are at least $n(n - 3) - \binom{n-3}{3}$ syzygies in the top-degree parts. For $n \leq 13$ this turns out to be more than 0. This explains why $n = 13$ runs so much faster than anticipated. With experiments, we were able to verify that these are indeed degree falls and not complete syzygies and that the number is correct. The results are given in Table 13.

Table 13. Structural degree falls in tri-degree $(1, 1, 1)$, experimentally verified.

n	degree falls
10	35
11	32
12	24
13	10

For $n \geq 14$ these degree falls do not appear anymore, but there are others. Let $g \in \mathcal{R}$ be a homogeneous polynomial in tri-degree $(d - 1, d - 1, d)$ such that it is symmetric in the first two arguments, i.e. $g(\mathbf{x}, \mathbf{y}, \mathbf{z}) = g(\mathbf{y}, \mathbf{x}, \mathbf{z})$. Now, due to the symmetry of g , $\overline{f_{\mathbf{xy}}^{(i)}} \cdot g$ is still alternating in its first two elements. Therefore, $\mathcal{S}(\overline{f_{\mathbf{xy}}^{(i)}} \cdot g)$ is again alternating and in the degree (d, d, d) Macaulay space. We

can apply rank-nullity again for the subspace generated by such g and find that the amount of top-degree syzygies appearing is

$$\binom{M_{n-3,d-1} + 1}{2} \cdot M_{n-3,d} \cdot n - \binom{M_{n-3,d}}{3}.$$

We did experiments to verify these numbers but found that there are even more degree falls appearing than predicted by the formula above. All of those were indeed degree falls and not full syzygies. Due to the size of the matrices involved only a part of the predictions could be verified, see Table 14.

Table 14. Structural degree falls in tri-degree (2, 2, 2) and (3, 3, 3).

degree falls (2, 2, 2)			degree falls (3, 3, 3)	
n	predicted	actual	n	predicted
14	15224	18941	17	23866080
15	15184	22337	20	77142736
16	11011		25	245964576

Even after identifying these (2, 2, 2) degree falls for $n = 15$ and adding those to the system, unfortunately, the system was still not solvable in total degree 5. So while these syzygies likely contribute to a lower solving degree than $11 = 5 + 5 + 1$ (as in Table 10), we do not (yet) know by how much.

6 Potential Generalizations and Future Work

Given the success of these small graph invariants on 3-TI and its structured variant, the question arises as to how this generalizes to other structures on tensors. A quick back-of-the-envelope calculation tells us the expectation value for similar structures in QMLE and Cubic-IP should be $1/q$ too. However, as we saw already in Remark 3, experimental evidence is needed before drawing conclusions.

Another interesting generalization might be to see how well this applies for k -tensors with $k \geq 4$. While these tensors are not (yet) cryptographically relevant, this might still provide insights into general invariants of tensors. One obvious hurdle in this case is that simple graphs might not be sufficient anymore.

Next to these generalizations, there are also two other open questions remaining after this work. It would be interesting to know how extra degree falls that we found contribute to the solving degree. Being able to make better predictions for the solving degree would give us more precise security estimates. Secondly, those degree falls also seem to exist, albeit fewer, in general systems of skew-symmetric bilinear equations. These might be used to speed up solving such systems as well.

A Lower Bounds on Probabilities of Triangles

The purpose of this section is to prove Corollary 1 and Corollary 2. To lower bound the probability of triangles we are going to use the following lemma.

Lemma 4. *Let X be a discrete distribution with values in $\mathbb{Z}_{\geq 0}$. Then:*

$$\mathbb{P}(X = 1) \geq 2\mathbb{E}(X) - \mathbb{E}(X^2).$$

Proof. Using that probabilities are non-negative:

$$\mathbb{P}(X = 1) \geq \sum_{i \geq 0} (2i - i^2) \mathbb{P}(X = i) = 2\mathbb{E}(X) - \mathbb{E}(X^2).$$

Now, using this lemma, the task at hand is to compute $\mathbb{E}(X^2)$ for MCE and ATFE. Let us denote $T\Delta\mathcal{C}$ (resp. $T\Delta\phi$) if T is a triangle for \mathcal{C} (resp. ϕ).

A.1 MCE

Lemma 5. *Suppose we have vector spaces U , V , and W over \mathbb{F}_q of dimensions n , m , and k , then:*

$$\begin{aligned} & \mathbb{E}_{\mathcal{C} \in (U \otimes V \otimes W)^*} (|\{T \in \mathbb{T}(U, V, W) \mid T\Delta\mathcal{C}\}|^2) \\ & \approx q^{-1} + q^{-2} + q^{-(n-1)} + q^{-(m-1)} + q^{-(k-1)} \\ & + q^{-(n+m-k)} + q^{-(m+k-n)} + q^{-(k+n-m)}. \end{aligned}$$

Proof. We start with the observation that, for a given tensor \mathcal{C} , we have:

$$|\{T \in \mathbb{T} \mid T\Delta\mathcal{C}\}|^2 = |\{(T_1, T_2) \in \mathbb{T}^2 \mid T_1\Delta\mathcal{C}, T_2\Delta\mathcal{C}\}|.$$

We are going to calculate the probability that both $T_1 = (u, v, w)$ and $T_2 = (u', v', w')$ are triangles for a random \mathcal{C} . To do this, we need to distinguish some cases. We define the following partition of \mathbb{T}^2 :

$$\begin{aligned} \mathbb{T}_\emptyset &= \{(T_1, T_2) \in \mathbb{T}^2 \mid u \neq u', v \neq v', w \neq w'\} \\ \mathbb{T}_u &= \{(T_1, T_2) \in \mathbb{T}^2 \mid u = u', v \neq v', w \neq w'\} \quad (\text{resp. } \mathbb{T}_v, \mathbb{T}_w) \\ \mathbb{T}_{uv} &= \{(T_1, T_2) \in \mathbb{T}^2 \mid u = u', v = v', w \neq w'\} \quad (\text{resp. } \mathbb{T}_{vw}, \mathbb{T}_{wu}) \\ \mathbb{T}_{uvw} &= \{(T_1, T_2) \in \mathbb{T}^2 \mid u = u', v = v', w = w'\}. \end{aligned}$$

For each of these sets, the probability that an element is a pair of triangles for a random \mathcal{C} is constant. To see this we are going to consider the tensor in the extended basis $\langle u, u_2, \dots, u_n \rangle$ if $u = u'$ and $\langle u, u', u_3, \dots, u_n \rangle$ if $u \neq u'$ (similarly for v and w). Then, the coefficients of these tensors are again uniformly random. We count the amount of indices (i, j, l) which should have a 0 so that T_1 and T_2 are triangles in Table 15.

Table 15. Sizes and indices that should be zero for the different sets in the partition.

Set	Size (\log_q)	Zeros	Set of index sets
\mathbb{T}_0	$6n - 18$	$6n - 16$	$\{1, 2, \star\}, \{1, 3, \star\}, \{2, 3, \star\}, \{4, 5, \star\}, \{4, 6, \star\}, \{5, 6, \star\}$
\mathbb{T}_1	$5n - 13$	$6n - 20$	$\{1, 2, \star\}, \{1, 3, \star\}, \{2, 3, \star\}, \{1, 4, \star\}, \{1, 5, \star\}, \{4, 5, \star\}$
\mathbb{T}_2	$4n - 10$	$5n - 14$	$\{1, 2, \star\}, \{1, 3, \star\}, \{2, 3, \star\}, \{1, 4, \star\}, \{2, 4, \star\}$
\mathbb{T}_3	$3n - 9$	$3n - 8$	$\{1, 2, \star\}, \{1, 3, \star\}, \{2, 3, \star\}$

Then, by linearity we can compute the expectation value:

$$\begin{aligned}
 \mathbb{E}(|\{T_1, T_2 \in \mathbb{T} \mid T_1 \Delta \mathcal{C}, T_2 \Delta \mathcal{C}\}|) &= q^{-(2n+2m+2k-4)} |\mathbb{T}_\emptyset| \\
 &+ q^{-(2n+2m+2k-6)} (|\mathbb{T}_u| + |\mathbb{T}_v| + |\mathbb{T}_w|) \\
 &+ q^{-(2n+2m+k-4)} |\mathbb{T}_{uv}| + q^{-(n+2m+2k-4)} |\mathbb{T}_{vw}| \\
 &+ q^{-(2n+m+2k-4)} |\mathbb{T}_{wu}| + q^{-(n+m+k-2)} |\mathbb{T}_{uvw}| \\
 &= q^{-1} + q^{-2} + q^{-(n-1)} + q^{-(m-1)} + q^{-(k-1)} \\
 &+ q^{-(n+m-k)} + q^{-(m+k-n)} + q^{-(k+n-m)}.
 \end{aligned}$$

Corollary 3. *Given vector spaces U , V , and W over \mathbb{F}_q of dimensions n , m , and k , then:*

$$\begin{aligned}
 \mathbb{P}_{\mathcal{C} \in (U \otimes V \otimes W)^*}(\mathcal{C} \text{ has unique triangle}) &\geq q^{-1} - q^{-2} - q^{-(n-1)} - q^{-(m-1)} - q^{-(k-1)} \\
 &- q^{-(n+m-k)} - q^{-(m+k-n)} - q^{-(k+n-m)}.
 \end{aligned}$$

Now Corollary 1 follows immediately.

A.2 ATFE

Lemma 6. *Suppose we have a vector space V over \mathbb{F}_q of dimension n then:*

$$\mathbb{E}_{\phi \in (\wedge^3 V)^*} (|\{T \in \mathbb{T}(V) \mid T \Delta \phi\}|^2) \approx q^{-1} + q^{-2} + q^{-(n-4)} + q^{-(n-7)}.$$

Proof. The proof is structured similarly. This time, we partition \mathbb{T}^2 in the following sets:

$$\mathbb{T}_i = \{(T_1, T_2) \in \mathbb{T}^2 \mid \dim(T_1 \cap T_2) = i\} \quad \text{for } i = 0, 1, 2, 3$$

The probability is again constant on these sets for random ϕ . Given a pair of triangles $(T_1, T_2) \in \mathbb{T}_i$ we pick a basis $\langle u_1, \dots, u_n \rangle$ such that, $T_1 = \langle u_1, u_2, u_3 \rangle$ and

$$T_2 = \begin{cases} \langle u_4, u_5, u_6 \rangle & \text{if } i = 0, \\ \langle u_1, u_4, u_5 \rangle & \text{if } i = 1, \\ \langle u_1, u_2, u_4 \rangle & \text{if } i = 2, \\ \langle u_1, u_2, u_3 \rangle & \text{if } i = 3. \end{cases}$$

Table 16. Sizes and indices that should be zero for the different sets in the partition.

Set	Size (\log_q)	Zeros	Set of index sets
\mathbb{T}_0	$6n - 18$	$6n - 16$	$\{1, 2, \star\}, \{1, 3, \star\}, \{2, 3, \star\}, \{4, 5, \star\}, \{4, 6, \star\}, \{5, 6, \star\}$
\mathbb{T}_1	$5n - 13$	$6n - 20$	$\{1, 2, \star\}, \{1, 3, \star\}, \{2, 3, \star\}, \{1, 4, \star\}, \{1, 5, \star\}, \{4, 5, \star\}$
\mathbb{T}_2	$4n - 10$	$5n - 14$	$\{1, 2, \star\}, \{1, 3, \star\}, \{2, 3, \star\}, \{1, 4, \star\}, \{2, 4, \star\}$
\mathbb{T}_3	$3n - 9$	$3n - 8$	$\{1, 2, \star\}, \{1, 3, \star\}, \{2, 3, \star\}$

Note that this is possible by first picking a basis of $T_1 \cap T_2$ by definition of \mathbb{T}_i . Then, the coefficients of these ATFs are again uniformly random. Now we count the number of index sets $\{i, j, k\}$ such that ϕ_{ijk} must be zero in order for T_1 and T_2 to be a triangle. These results are in table Table 16

Then, again by linearity, we can compute the expectation value:

$$\begin{aligned} \mathbb{E}(|\{T_1, T_2 \in \mathbb{T} \mid T_1 \triangle \phi, T_2 \triangle \phi\}|) \\ &= q^{-(6n-16)}|\mathbb{T}_0| + q^{-(6n-20)}|\mathbb{T}_1| + q^{-(5n-14)}|\mathbb{T}_2| + q^{-(3n-8)}|\mathbb{T}_3| \\ &= q^{-1} + q^{-2} + q^{-(n-7)} + q^{-(n-4)}. \end{aligned}$$

Corollary 4. *Given a vector space V over \mathbb{F}_q of dimension $n = \dim(V)$ then:*

$$\mathbb{P}_{\phi \in (\wedge^3 V)^*}(\phi \text{ has a unique triangle}) \geq q^{-1} - q^{-2} - q^{-(n-7)} - q^{-(n-4)}.$$

Now Corollary 2 follows immediately.

References

1. NIST fourth round announcement. NIST Official Website (2021), <https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions>
2. Bardet, M., Faugère, J., Salvy, B., Spaenlehauer, P.: On the complexity of solving quadratic Boolean systems. *Journal of Complexity* **29**(1), 53–75 (2013)
3. Barenghi, A., Biasse, J., Persichetti, E., Santini, P.: LESS-FM: fine-tuning signatures from the code equivalence problem. In: Cheon, J.H., Tillich, J. (eds.) PQCrypto 2021. LNCS, vol. 12841, pp. 23–43. Springer (2021)
4. Beullens, W.: Not enough LESS: an improved algorithm for solving code equivalence problems over \mathbb{F}_q . In: Dunkelman, O., Jacobson, M.J., O’Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 387–403. Springer (2020)
5. Beullens, W.: Graph-theoretic algorithms for the alternating trilinear form equivalence problem. In: Handschuh, H., Lysyanskaya, A. (eds.) *Advances in Cryptology - CRYPTO 2023*, pp. 101–126. Springer Nature Switzerland, Cham (2023)
6. Beullens, W., Katsumata, S., Pintore, F.: Calamari and Falaf: Logarithmic (linkable) ring signatures from isogenies and lattices. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 464–492. Springer (2020)
7. Biasse, J.F., Micheli, G., Persichetti, E., Santini, P.: LESS is More: Code-Based Signatures Without Syndromes. In: Nitaj, A., Youssef, A. (eds.) AFRICACRYPT 2020. LNCS, vol. 12174, pp. 45–65. Springer (2020)

8. Bläser, M., Duong, D.H., Narayanan, A.K., Plantard, T., Qiao, Y., Sipasseuth, A., , Tang, G.: The ALTEQ Signature Scheme: Algorithm Specifications and Supporting Documentation. NIST PQC Submission (2023)
9. Bosma, W., Cannon, J., Playoust, C.: The Magma Algebra System. I. The User Language. *J. Symbolic Comput.* **24**(3-4), 235–265 (1997)
10. Bouillaguet, C., Fouque, P., Véber, A.: Graph-theoretic algorithms for the “isomorphism of polynomials” problem. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 211–227. Springer (2013)
11. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.D. thesis, University of Innsbruck (1965)
12. Chou, T., Niederhagen, R., Persichetti, E., Ran, L., Randrianarisoa, T.H., Reijnders, K., Samardjiska, S., Trimoska, M.: MEDS – Matrix Equivalence Digital Signature (2023), <https://meds-pqc.org/spec/MEDS-2023-05-31.pdf>, submission to the NIST Digital Signature Scheme standardization process
13. Chou, T., Niederhagen, R., Persichetti, E., Randrianarisoa, T.H., Reijnders, K., Samardjiska, S., Trimoska, M.: Take your meds: Digital signatures from matrix code equivalence. In: El Mrabet, N., De Feo, L., Duquesne, S. (eds.) Progress in Cryptology - AFRICACRYPT 2023, pp. 28–52. Springer Nature Switzerland, Cham (2023)
14. Courtois, N.T., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer (2000)
15. Couvreur, A., Debris-Alazard, T., Gaborit, P.: On the hardness of code equivalence problems in rank metric. arXiv (2021)
16. De Feo, L., Galbraith, S.D.: SeaSign: Compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 759–789. Springer (2019)
17. Draisma, J., Shaw, R.: Singular lines of trilinear forms. *Linear algebra and its applications* **433**(3), 690–697 (2010)
18. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra* **139**, 61–88 (1999)
19. Faugère, J.C., Perret, L.: Polynomial equivalence problems: Algorithmic and theoretical aspects. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 30–47. Springer (2006)
20. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM* **38**(3), 690–728 (jul 1991). <https://doi.org/10.1145/116825.116852>, <https://doi.org/10.1145/116825.116852>
21. Grochow, J.A., Qiao, Y.: Isomorphism problems for tensors, groups, and cubic forms: completeness and reductions (2019)
22. Grochow, J.A., Qiao, Y., Tang, G.: Average-case algorithms for testing isomorphism of polynomials, algebras, and multilinear forms. *Journal of Groups, Complexity, Cryptology* **Volume 14, Issue 1** (Aug 2022). <https://doi.org/10.46298/jgcc.2022.14.1.9431>, <https://gcc.episciences.org/9836>, preliminary version appeared in STACS ’21. <https://doi.org/10.4230/LIPIcs.STACS.2021.38>. Preprint available at [arXiv:2012.01085](https://arxiv.org/abs/2012.01085)
23. Hulsing, A., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Kampanakis, P., Kolbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Aumasson, J.P., Westerbaan, B., Beullens, W.: SPHINCS+. NIST PQC Submission (2020)

24. Hülsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: extended hash-based signatures. RFC 8391 (2018)
25. ISO (International Organization for Standardization): Information security, cybersecurity and privacy protection: Iso/iec wd 14888-4 information technology - security techniques - digital signatures with appendix - part 4: Stateful hash-based mechanisms, <https://www.iso.org/standard/80492.html>
26. Joux, A., Vitse, V.: A Crossbred Algorithm for Solving Boolean Polynomial Systems. In: Kaczorowski, J., Pieprzyk, J., Pomykała, J. (eds.) *Number-Theoretic Methods in Cryptology*, pp. 3–21. Springer International Publishing, Cham (2018)
27. Lazard, D.: Gröbner-Bases, Gaussian elimination and resolution of systems of algebraic equations. In: van Hulzen, J.A. (ed.) *EUROCAL. Lecture Notes in Computer Science*, vol. 162, pp. 146–156. Springer (1983)
28. Leon, J.S.: Computing automorphism groups of error-correcting codes. *IEEE Trans. Inf. Theory* **28**(3), 496–510 (1982)
29. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. NIST PQC Submission (2020)
30. Narayanan, A.K., Qiao, Y., Tang, G.: Algorithms for matrix code and alternating trilinear form equivalences via new isomorphism invariants. Springer-Verlag (2024)
31. NIST (National Institute for Standards and Technology): Post-Quantum Cryptography Standardization (2017). <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography>
32. NIST (National Institute for Standards and Technology): Fifth PQC Standardization Conference (2024). <https://csrc.nist.gov/Events/2024/fifth-pqc-standardization-conference>
33. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms. In: *EUROCRYPT '96. LNCS*, vol. 1070, pp. 33–48. Springer (1996)
34. Perret, L.: On the computational complexity of some equivalence problems of polynomial systems of equations over finite fields. *Electronic Colloquium on Computational Complexity (ECCC)* (116) (2004)
35. Perret, L.: A Fast Cryptanalysis of the Isomorphism of Polynomials with One Secret Problem. In: *EUROCRYPT. Lecture Notes in Computer Science*, vol. 3494, pp. 354–370. Springer (2005)
36. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. NIST PQC Submission (2020)
37. Ran, L., Samardjiska, S., Trimoska, M.: Algebraic algorithm for the alternating trilinear form equivalence problem. In: Esser, A., Santini, P. (eds.) *Code-Based Cryptography*, pp. 84–103. Springer Nature Switzerland, Cham (2023)
38. Reijnders, K., Samardjiska, S., Trimoska, M.: Hardness estimates of the code equivalence problem in the rank metric. *Designs, Codes and Cryptography* **92**, 1–30 (01 2024). <https://doi.org/10.1007/s10623-023-01338-x>
39. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D.: CRYSTALS-KYBER. NIST PQC Submission (2020)
40. Tang, G., Duong, D.H., Joux, A., Plantard, T., Qiao, Y., Susilo, W.: Practical post-quantum signature schemes from isomorphism problems of trilinear forms. In: *EUROCRYPT 2022. LNCS*, vol. 13277, pp. 582–612. Springer (2022)

Fault Attacks and Side-Channel Analysis



It's a Kind of Magic: A Novel Conditional GAN Framework for Efficient Profiling Side-Channel Analysis

Sengim Karayalçın¹(✉) , Marina Krček² , Lichao Wu³ , Stjepan Picek⁴ ,
and Guilherme Perin¹ 

¹ Leiden University, Leiden, Netherlands

{s.karayalcin,g.perin}@liacs.leidenuniv.nl

² Delft University of Technology, Delft, Netherlands

m.krcek@tudelft.nl

³ Technical University of Darmstadt, Darmstadt, Germany

lichao.wu@tu-darmstadt.de

⁴ Radboud University, Nijmegen, Netherlands

stjepan.picek@ru.nl

Abstract. Profiling side-channel analysis (SCA) is widely used to evaluate the security of cryptographic implementations under worst-case attack scenarios. This method assumes a strong adversary with a fully controlled device clone, known as a profiling device, with full access to the internal state of the target algorithm, including the mask shares. However, acquiring such a profiling device in the real world is challenging, as secure products enforce strong life cycle protection, particularly on devices that allow the user partial (e.g., debug mode) or full (e.g., test mode) control. This enforcement restricts access to profiling devices, significantly reducing the effectiveness of profiling SCA.

To address this limitation, this paper introduces a novel framework that allows an attacker to create and learn from their own white-box reference design without needing privileged access on the profiling device. Specifically, the attacker first implements the target algorithm on a different type of device with full control. Since this device is a white box to the attacker, they can access all internal states and mask shares. A novel conditional generative adversarial network (CGAN) framework is then introduced to mimic the feature extraction procedure from the reference device and transfer this experience to extract high-order leakages from the target device. These extracted features then serve as inputs for profiled SCA. Experiments show that our approach significantly enhances the efficacy of black-box profiling SCA, matching or potentially exceeding the results of worst-case security evaluations. Compared with conventional profiling SCA, which has strict requirements on the profiling device, our framework relaxes this threat model and, thus, can be better adapted to real-world attacks.

1 Introduction

Commonly used cryptographic algorithms, such as AES and 3DES, are mathematically secure, as simply knowing the input and output data along with the details of the algorithm is insufficient to recover the key within a reasonable computation time. However, cryptographic implementations on hardware may introduce unintentional information leakages via, for instance, power consumption [27], electromagnetic emission (EM) [1], execution time [26], temperature [23], and acoustics [17]. These leakages could be exploited through side-channel analysis (SCA) and finally extract secret information.

The research community has known the threat of SCA for more than 25 years. Multiple attacks have been developed and can be generally categorized into two groups, depending on the availability of the profiling device. Non-profiling SCA leverages statistical methods, distinguishers, and leakage assessment techniques to launch direct side-channel attacks. Examples of such attacks include Differential Power Analysis [28], Correlation Power Analysis [5], and Mutual Information Analysis [18]. These attack methods are implicitly treated as real-world security threats, mainly because an attack is directly mounted on the victim’s device; querying encryption or decryption executions are the only requirements to deploy the attack. On the other hand, research on profiling SCA, such as template attack [11] and deep learning-based SCA [34,45] has largely aimed at enabling worst-case security assessments [7] with an assumption of access to an identical copy of the target device. A profiling model is first built by mapping the relationship between the leakage measurements and the corresponding labels (key-related intermediate data) obtained from this copy. Then, an attacker collects leakage measurements from the target device and feeds them to the profiling model to obtain the label prediction. On top of this threat model, recent works further assume insight into values of mask shares generated during cryptographic executions [36], making the attack fully white-box. An attacker can profile each intermediate data or mask share and can finally combine the profiling results together to recover the secret. Although optimal attack performance can be reached via this threat model, we argue this threat model, including the access to an identical device copy and the mask-shares knowledge, is not realistic, as secure products often enforce robust life cycle protections, especially on devices that offer partial (e.g., debug mode) or full (e.g., test mode) user control. Even for the security evaluation labs that are supposed to have all design details [42], this threat model is overly strong, as the mask shares are commonly stored in protected registers and are not accessible even by a kernel user. Even in evaluation settings for software implementations, it is often not possible to access randomness as this would require modifications to the implementation (e.g., using a known seed or instrumenting source code), which results in the evaluation targeting a characterization of the implementation as opposed to the actual target [30]. On the other hand, if an attack can be performed with this attack assumption, attacks are significantly easier as an attacker can profile each individual mask share and finally recover the secret with, for instance, Soft Analytical Side-Channel Attacks [40].

This paper introduces a novel framework based on conditional generative adversarial networks (CGAN) to address the overly strong attack assumptions in profiling side-channel analysis (SCA). In line with [30], we assume that the attacker is aware of the cryptographic implementation and masking scheme used on the target device. With this design knowledge, the attacker sets up a similar cryptographic implementation on a *different type* of the device, referred to as the *reference implementation*. Since the attacker is not restricted by the device type, they can freely choose devices that grant full control and access to all internal states of the target algorithm, including mask shares. The proposed CGAN-based structure is then introduced to mimic white-box feature selection performed on the reference implementation and efficiently extract features from a target implementation with unknown masks. This framework transforms a conventional (black-box) profiling attack into a white-box profiling attack but with reduced attack assumptions. Additionally, our framework enhances the interpretability of the attack on the target dataset by splitting the feature extraction and exploitation phases, providing deeper insights into the attack process.

In summary, our main contributions are:

- We propose a novel conditional generative adversarial network-based SCA framework (CGAN-SCA) that allows an adversary to leverage the knowledge from a reference implementation to extract features from a target implementation. This modified threat model and corresponding CGAN-based framework demonstrate the potential risks that arise when an adversary has full control over an implementation similar to the target one.
- The proposed framework allows an adversary to convert a black-box profiling attack towards a white-box profiling attack capability, which drastically improves the black-box profiling attack performance. Our results demonstrate that applying our framework significantly reduces the difficulties of finding an optimal profiling model in a non-worst-case security evaluation.
- The proposed CGAN-SCA framework can extract features from high-order leakages, such as first-order masking schemes. We provide a detailed analysis to demonstrate how the generator in a CGAN architecture precisely mimics the features selected from a reference implementation.
- Our results indicate that once an efficient CGAN architecture is found, a hyperparameter search for a profiling attack can be done with negligible effort, similar to a worst-case security evaluation.¹

Note: More experimental results can be found in [25]. Our source code is available in an anonymous repository.²

¹ We refer to Section 6 of [30] for a discussion about difficulties in finding deep learning-based profiling models in worst and non-worst case security evaluations.

² https://anonymous.4open.science/r/cgan_sca-7A33/.

2 Background

2.1 GANs and CGANs

Generative models are machine learning models that learn the underlying probability distribution of a given dataset [19]. Their primary objective is to generate new samples that resemble the training data in terms of statistical properties and structure. While discriminative models focus on learning the decision boundary between different classes or categories of data, generative models aim to understand and capture the characteristics and patterns of the entire dataset.

Generative adversarial networks proposed a novel way to train generative models [20]. The overall idea is to adversarially train a generator and discriminator where the discriminator attempts to differentiate between real and generated images, and the generator is trained to generate fake images that fool the discriminator. These types of models have been used extensively across a wide variety of domains. Examples include image generation [20], image translation [24], and speech-synthesis [29].

As shown in Fig. 1a, the structure consists of two adversarial models competing against each other: a generator \mathcal{G} , with parameters θ_g , and a discriminator \mathcal{D} , with parameters θ_d . The main goal of the generator is to take input noise distribution $p(z)$ and to produce synthetic or fake output data $\mathcal{G}(z, \theta_g)$ that follows a data distribution present in real data. The discriminator is trained to provide the probability $\mathcal{D}(x, \theta_d)$ that an input data x comes from a real training set or the generator. Both generator and discriminator are trained simultaneously in a way that θ_g to minimize $\log(1 - \mathcal{D}(\mathcal{G}(z)))$ and θ_d to minimize $\log \mathcal{D}(x)$, as following a min-max game with value function:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{x \sim p(x)}[\log \mathcal{D}(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - \mathcal{D}(\mathcal{G}(z)))]. \quad (1)$$

Conditional Generative Adversarial Networks (CGANs) [32], illustrated in Fig. 1b, are a variant of the traditional GAN architecture incorporating additional information to guide the generation process. In CGANs, the generator and discriminator receive extra input in the form of conditional variables, which can be class labels, attribute vectors, or any other auxiliary information. This conditioning allows for generating more targeted and controlled outputs.

2.2 Generative Models for SCA

Generative models in side-channel analysis have been limited to a few applications. In [41], the authors considered generative adversarial networks for data augmentation. Later, a more elaborated analysis with conditional generative adversarial networks also considered data augmentation [33]. Both analyses were applied to protected AES implementations. In [44], the authors considered Variational AutoEncoders (VAE) to generate reconstructed and synthetic traces that model the true conditional probability distribution of real leakage traces. In [13], the authors proposed the EVIL-machine, a framework using a GAN-like structure to find a suitable leakage model for the target device, replacing the need

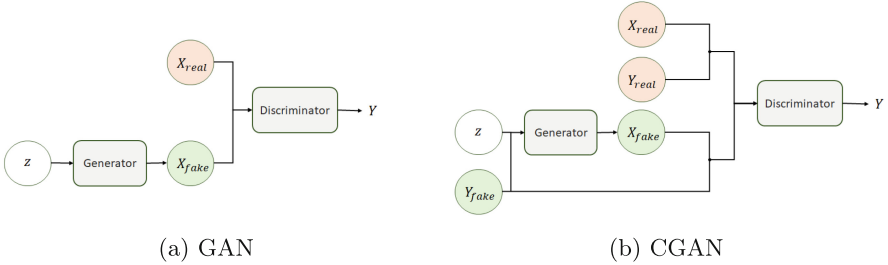


Fig. 1. GAN and CGAN structures.

for prior knowledge of the leakage characteristics. The structure is extended to mount non-profiled attacks that exploit the learned leakage model. In [10], the authors presented an approach using a GAN-based structure to mitigate the issues related to the portability of profiling models. Like our framework, the authors extracted an intermediate representation of the leakages from a profiling device and then trained a generator to extract a similar representation from unlabeled attack traces measured on another device of the same model. They considered only unprotected implementations running on the same device model. Finally, in [16], a GAN structure is used to translate between side-channel domains. To accomplish this, pairs of traces are required. Again, only unprotected implementations are considered. While these works consider GAN structures to transform leakage traces, these require paired measurements in different side-channel domains [16] or only consider portability [10]. As such, the differences between the adversarial and target datasets are fairly minimal, allowing for training the GAN structure without labels. In this work, the differences are more significant, necessitating the inclusion of labels in the discriminator to facilitate the convergence of our models.

When we look at applications in DLSCA where information on other implementations is utilized to build more powerful models, the use cases are still limited. Several works have looked at utilizing transfer learning techniques to limit the profiling complexity of attacking novel targets [15, 38, 43]. These works generally look at fine-tuning models that were pre-trained on a similar task, which reduces the number of profiling traces required from the profiling device. Similarly, several works incorporate knowledge of the masking scheme to implement tailored DL layers that explicitly recombine secret shares [9, 30]. The main benefits here are, again, to reduce the number of required profiling traces. The CGAN-SCA framework that we propose in the next section acts as a feature extractor from raw datasets, and therefore, our work can also be seen as a pre-processing method. Regarding applying deep neural networks for preprocessing leakage traces specifically, we refer to Section 4.2 from [36]. We emphasize that none of the related works consider a generative adversarial architecture to efficiently extract features from a target dataset by learning the probability distribution from an adversarial dataset, as detailed next.

2.3 Signal-to-Noise Ratio (SNR)

The signal-to-noise ratio (SNR) measures the strength of a desired signal compared to the background noise level, indicating the clarity and quality of the signal in relation to the interference or irrelevant information present. In the side-channel analysis context, SNR is a measure of the amount of leakage that is present in a side-channel trace. It is defined as:

$$SNR = \frac{Var(\mathbf{E}(X|Y))}{\mathbf{E}(Var(X|Y))}. \quad (2)$$

Here, \mathbf{E} represents the arithmetic mean function and Var is the variance. Generally, we assume X to be a single point in a side-channel trace, and we condition on Y , representing the intermediate value leaked.

3 CGAN-SCA Framework

This section proposes a novel profiling attack framework based on conditional generative adversarial networks for side-channel analysis (CGAN-SCA). To this end, we propose an extended profiling SCA threat model, as described next.

3.1 Threat Model and Notations

The classic threat model for profiled attacks in SCA assumes an attacker has an open and programmable copy of the attacked device [11]. With this privileged access right, an attacker can enable and disable countermeasures and has access to mask shares used to generate masks (on the profiling device). This threat model allows an evaluator to create templates for each secret share straightforwardly and then explicitly recombine the retrieved shares during the attack phase. As a consequence, this attack assumption leads to near-optimal attack performance with relatively limited resources [8]. However, as mentioned in the introduction, access to mask shares is often not possible in practical settings. In line with the scheme-aware threat model proposed in [30], where an attacker has (some) knowledge of the implementation specifics of the masking scheme but does not have access to mask shares, this work extends this threat model by including an additional device (on top of the profiling and attack device) that runs a similar implementation in a fully white-box setting (i.e., with access to mask values during profiling). We refer to this device as the reference implementation. Here, access to mask values is not a problem as the reference is a separate implementation the adversary develops themselves. Therefore, any restrictions that apply to the identical device clone that is used in conventional profiling SCA do not apply to the reference implementation. In scenarios where an attacker has access to the target source code, this can even be used to create the reference dataset by instrumenting it and taking separate measurements. In less permissive scenarios where an attacker only knows the type of countermeasure (i.e., the target is protected with first-order Boolean masking), they can create a different

implementation that runs the same (type of) countermeasure. If a similar public dataset is available, the reference implementation could even be a publicly released dataset.

One may doubt the similarity between the reference device and the target device. It is intuitively clear that the more similar the reference and target device are, the better. However, when differences are too large, the framework might not be able to improve over standard profiling attacks. Broadly, it is advisable to take devices (and implementations) that operate on the same word sizes. In practice, this means that it seems unlikely that using a software device that operates on 8 bits of the internal state at a time as a reference can help when we are trying to attack a hardware (or bit-sliced) implementation that operates on larger states. Based on our results, improvements to black-box attacks can be achieved even when reference and target are running on different device architectures and running different implementations of the same masking scheme. A more in-depth discussion can be found in Sect. 8.

Based on our threat model, we refer to three categories of trace sets: X_{ref} , X_{prof} , and X_{target} , representing leakages from the reference, profiling, and target devices, respectively. For clarity, X_{ref} has known key(s) and masks, X_{prof} has known key(s) and unknown masks, and X_{target} has unknown key and masks. A feature selection process over X_{ref} results in an adversarial dataset³ for the proposed CGAN, also referred to as reference features f_{ref} . The features extracted with the proposed CGAN-based architecture from X_{prof} and X_{target} are referred to as target or generated features f_{prof} and f_{target} . N_f is the number of features in the f_{ref} , f_{prof} , or f_{target} sets. The validation set is a subset of X_{prof} , denoted as X_{val} with features f_{val} . Note that the validation set is excluded from training and used to simulate attacks for hyperparameter tuning.

3.2 A Novel Conditional GAN Framework

The proposed Conditional GAN-based framework, referred to as CGAN-SCA, is illustrated in Fig. 2. The structure consists of the following main blocks:

1. Feature selection (top of Fig. 2): this block receives at its inputs the set of reference side-channel measurements X_{ref} and the masks (randomness) associated with this dataset. This block outputs the features f_{ref} (i.e., the adversarial dataset), which should contain the most leaky samples from X_{ref} , similar to the points of interest (POI) selection. In this paper, we consider different methods for feature selection: SNR, Linear Discriminant Analysis (LDA), and Principal Component Analysis (PCA). The feature selection process must only be done once for each reference dataset and feature selection method.
2. Generator \mathcal{G} (middle of Fig. 2): this block receives the side-channel measurements from the target implementation at its inputs, X_{prof} or X_{target} . The generator’s output is the set of extracted features, f_{prof} or f_{target} , also a

³ The term *adversarial* is not connected with the domain of security of AI, e.g., adversarial examples, but with the fact that it is a dataset used by an adversary utilizing a GAN.

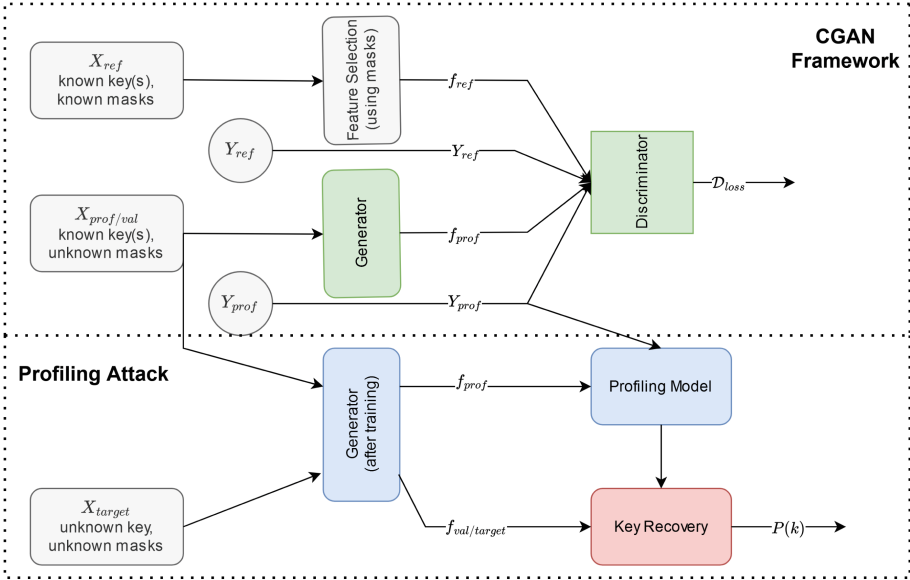


Fig. 2. Proposed CGAN-SCA framework. X_{ref} is the reference dataset with known secrets (i.e., masks and keys), X_{prof} is the profiling set with known key(s) and unknown masks, and X_{target} is the target device with unknown key and mask. Y_{ref} and Y_{prof} are corresponding labels to the X_{ref} and X_{prof} datasets, respectively.

latent representation of the traces. It is trained to generate f_{prof} that looks real to the discriminator.

3. Discriminator \mathcal{D} (upper right of Fig. 2): this block receives at its input the set of features (f_{ref} or f_{prof}) and the corresponding set of labels (Y_{ref} or Y_{prof}). The output of the discriminator is a value representation of the loss function. The discriminator is trained to discriminate between real and generated features (f_{ref} and f_{prof}) using labels Y_{ref} and Y_{prof} .
4. Profiling attack (lower half of Fig. 2): after the generator \mathcal{G} is trained, it is used to generate f_{prof} and f_{target}/f_{val} and a profiling attack is applied on these features. The attack follows the same classic profiling attack structure (i.e., profiling and attack phases), where any type of profiling model can be used. The main difference is that the model is profiled with extracted features f_{prof} to attack f_{target}/f_{val} , which should contain only leaky points of interest from the original target traces.

The main goal of training the proposed CGAN model is to generate f_{target} outputs with the same dimension as given by f_{ref} and with most of its features containing main side-channel leakages from X_{target} . Thus, the generator acts as a feature extraction or dimensionality reduction mechanism. Different from a classic CGAN structure where the generator receives at its inputs a random source and a label, *our generator receives only the original traces X_{target} from the target implementation and no labels*. This is important as for X_{target} , we do not

have labels Y_{target} , and if the generator relies on labels for extracting features, it is not possible to apply the generator to X_{target} as labels are unavailable. We emphasize that this architecture is a new concept introduced in this paper.

The main goal in training the generator \mathcal{G} is to learn parameters $\theta_{\mathcal{G}}$ such that new samples f_{prof} are statistically indistinguishable from samples from reference f_{ref} . In other words, we train $\theta_{\mathcal{G}}$ to transform input target traces X_{target} to the probability distribution of f_{ref} . Determining the distance between two distributions is a two-sample hypothesis test problem, which is difficult for complicated distributions with high dimensions. Therefore, we will also judge the quality of the generator by computing the SNR between f_{target} (i.e., the output of the generator) and the high-order secret shares from the target device. In our threat model, an attacker does not know these high-order secret shares from X_{target} , and these values will not interfere with the training of generator and discriminator models. Here, we consider them only to visually confirm that the generator extracts meaningful representations from X_{target} . It is important to note that, during the CGAN training, we can only use X_{prof} , as the structure requires the knowledge of its labels Y_{prof} . In this paper, the labels Y_{ref} and Y_{prof} are always the value of the S-box output byte from the first AES encryption round, without assuming the knowledge of any mask value.

After training the CGAN structure, the trained generator model is used to transform the profiling, validation, and attack sets from X_{prof} and X_{target} into f_{prof} and f_{target} , as shown in the bottom part of Fig. 2. Note how this feature extraction process (i.e., predicting f_{target} from X_{target}) does not involve any label. In the next phase, we utilize the transformed f_{prof} and f_{target} sets to launch standard profiling attacks. In the attack phase, we obtain the probability $P(k)$ for each key candidate k , which allows us to derive the guessing entropy or success rate [37] of the correct key. Therefore, the main advantage of using the CGAN-SCA framework as a preprocessing step is that feature extraction can be done against a black-box profiling target, allowing key recovery results closer to white-box profiling attack performance without expensive hyperparameter tuning efforts, as shown in Sect. 7.

3.3 CGAN Architecture

We conducted preliminary experiments on the CGAN-SCA framework to determine well-performing (though not yet optimal) architectures for both the discriminator and generator models. Since this work only addresses synchronized datasets, we verified that small MLP-based architectures for the discriminator and generator already demonstrate satisfactory performance. However, a thorough hyperparameter search is essential to achieve better results. In the case of desynchronized measurements, CNN-based layers are highly recommended, but this is beyond the scope of this paper and will be explored in future works. In this section, we first describe the architectural choices for the discriminator and generator. Then, we discuss how to evaluate the efficacy of CGAN feature selection. We cover the specifics of our hyperparameter searches to find optimal solutions in Sect. 4.2.

During the training of the CGAN model, the objective of the discriminator is to distinguish between f_{ref} and f_{prof} . Conversely, the objective of the generator is to generate f_{prof} that is similar to f_{ref} . While these objectives will result in realistic-looking f_{prof} , the generator is not forced to extract the side-channel leakages from X_{prof} in any way as it is not conditioned. While a conventional CGAN model, where labels Y_{prof} are provided to both the generator and the discriminator, seems like a straightforward solution to alleviate this problem, the labels are unavailable during the attack phase. In other words, the generator needs to convert X_{target} into f_{target} without labels. As such, we provide labels only to the discriminator, which only received f_{prof} . This choice allows the discriminator to check whether the provided leakages in f_{prof} correspond to the label Y_{prof} . This will then force the generator to use the side-channel leakages in X_{prof} in its generated f_{prof} as otherwise, the discriminator can easily classify f_{prof} as fake.

Discriminator Architecture. We first look at how to construct the discriminator model as a poorly configured discriminator will always result in the CGAN model failing to generate useful f_{target} . Our main goal in constructing the discriminator is to ensure it uses the leakages in f_{ref} and f_{prof} and does not ‘memorize’ the correct f_{ref} . Several works have shown the capability of MLPs to learn to classify first-order protected datasets from relatively small intervals containing leaky samples [3] or even raw traces [34]. Thus, it should be relatively easy for an MLP-based discriminator to learn to combine leakages when its inputs contain only leaky samples. Developing architectures for other schemes should also be straightforward, as full access to secret shares of the reference implementation is available. Pre-training (part of) the discriminator in a classification task, as is done in [10], can also be an option. Learning higher-order schemes can then be accomplished using knowledge of secret shares during training [14, 31].

The discriminator serves two primary purposes: (1) classifying the input, which comprises a combination of labels and features, into two classes (0 or ‘fake’ and 1 or ‘real’), and (2) comprehending the relationship between labels and features. In the second case, we expect the discriminator to recognize an input combination of labels and features as ‘real’ if the features represent the corresponding label class. If the discriminator cannot classify whether a given combination of features and labels is real or fake, we assume that the generated features, denoted as f_{prof} , are as realistic as the reference features f_{ref} . The discriminator model is set with a binary cross-entropy loss function.

The number of features in f_{ref} and f_{prof} is limited to a maximum of $N_f = 100$, as the evaluated datasets contain a limited number of leaky points of interest to what concerns the processing of high-order leakages (e.g., masks and masked S-box output bytes). In the first experiments from Sect. 4, we define $N_f = 100$ for ASCADr, ASCADf, and DPAv4.2. For CHES CTF 2018 and ESHARD-AES128, we consider $N_f = 20$, as these two datasets are more noisy than previous ones.

Figure 3 illustrates the generic structure of the MLP-based discriminator architecture. The input label (due to the conditioned fashion of the CGAN

structure) is concatenated with the input features that can be either f_{ref} or f_{prof} . For this architecture, we use relatively large, fully connected (dense) layers after the embedding layer of the class label. Later, in Sect. 4, we refer to the number of dense layers after the embedding layer as *dense layers embedding*, in which the number of neurons in these layers will be referred to as *neurons embedding*. After the concatenation layer, we consider dense layers, and each one of them is followed by a dropout layer. Similarly to the embedding layers, the number of dense layers after the concatenation, whose are always interleaved with a dropout layer, will be referred to as *dense layers dropout*, each one with a number of neurons referred to as *neurons dropout*. The output layer of the discriminator always employs the *sigmoid* activation function for binary classification. Dropout layers are included in the discriminator as a means of regularization. We recommended performing hyperparameter tuning, using random search [34] as detailed in Sect. 4.2, to determine the optimal number of dense layers, their activation functions, and the corresponding number of neurons. To reduce the search space, this model utilizes the Adam optimizer with a learning rate of 0.0025 and a β value of 0.5. These hyperparameters are commonly employed in MLP-based profiling attacks [3, 36], and we assume they will also yield favorable results in this case. We emphasize that tuning is performed for the rest of the hyperparameters.

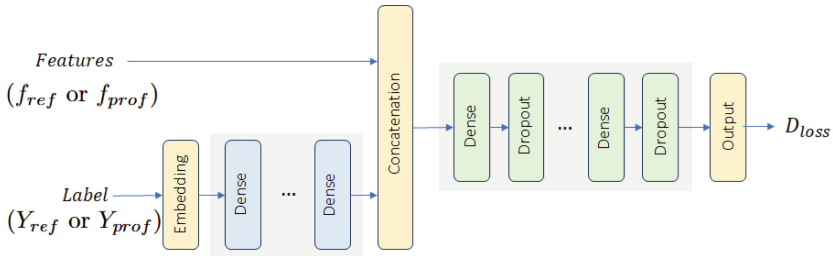


Fig. 3. Generic architecture for the discriminator.

Generator Architecture. Different from the originally proposed CGAN structure [32] and its variants [12, 46], our generator receives at its input real data X_{prof}/X_{target} rather than a noise distribution $p(z)$. The generator architecture is a simple MLP structure without any regularization mechanism. What is expected from the generator is to learn a mapping function $f(x, \theta_G) : X_{target} \rightarrow f_{target}$ representing a feature extraction process. When X_{target} is a set of leakage traces collected from a first-order masked AES implementation, the generator is expected to transfer from the input to the output the features from X_{target} that contain the highest SNR values with respect to two secret shares in the case of the first order masked dataset.

While the task the generator needs to perform is conceptually fairly simple, in practice, learning to extract leaky points of interest can be difficult. This is especially true when attacks against the (resampled) full-length traces are considered. In Table 9 of [34], we see that only between 0% and 5% of random models result in successful attacks against full-length traces, while when features are selected based on SNR values in the white-box scenario, almost all of them can successfully recover the target key byte. As such, finding an architecture that is well-tuned to the task of extracting these features also requires hyperparameter tuning effort.

3.4 Assessing CGAN’s Efficiency

Our CGAN-SCA framework assumes that only the reference device is fully controlled and that its secret shares are known. On the other hand, the randomness used to generate masks of the profiling/target dataset is unknown. This creates a challenging situation where accurately verifying the quality of the extracted features from X_{prof}/X_{target} becomes difficult. In simpler terms, we aim to measure the extent to which f_{target} represents the extracted high-order leakages from X_{target} when the target is an n -order masked implementation. To demonstrate the effectiveness of our CGAN-SCA solution, we utilize publicly available AES-128 datasets that also provide access to masks. Consequently, we calculate the SNR of the secret shares derived from the extracted features, f_{target} , which comes from the generator’s output. These SNR values are computed solely to confirm that the trained generator can automatically extract leakages from X_{target} . We emphasize that the CGAN model is neither trained nor validated using any information regarding the masks associated with the target dataset. Thus, for the targeted implementation, the threat model always follows the classic black-box profiling attack scenario.

At the end of each CGAN training epoch, we predict the generator with the attack set from the target dataset X_{target} , and we compute the SNR between extracted features f_{target} and the secret shares. This gives us two vectors with the same number of features from f_{target} . From these SNR vectors, we store the maximum SNR value. As the results from Sect. 4 confirm, the generator can extract features from X_{target} , and the SNR values of secret masks from f_{target} are high.

4 Experimental Results

This section first introduces the reference implementation we considered in this paper. Then, we perform a hyperparameter search to find generator and discriminator architectures for different reference and target dataset combinations. The best CGAN architectures are used to conduct profiling attacks and compare them with the state-of-the-art.

4.1 Datasets

Our framework requires limited similarity between the reference and target implementations. To illustrate this, this paper considers five publicly available AES software implementations and each of them can serve as a reference implementation. The implementation details and side-channel measurement setup are detailed in Table 1. The AES is implemented on different platforms with different instruction set architectures and clock speeds. In terms of leakage measurement, besides the difference in the leakage sources, the side-channel acquisition process varies significantly between each implementation: ASCAD datasets were acquired with a sampling rate of 2G samples per second (S/s), the ESHARD-AES128 dataset was measured with a sampling rate of 200MS/s (for other datasets, this information is not available).

Table 1. Dataset setups. All the datasets implement the AES-128 algorithm.

Dataset	Side-Channel Type	Platform and ISA	Clock Speed	Countermeasure
ASCADf [3]	EM	AVR RISC (8 bits)	4MHz	Boolean Masking
ASCADr [3]	EM	AVR RISC (8 bits)	4MHz	Boolean Masking
DPAv4.2 [4]	Power	AVR MIPS (8 bits)	4MHz	RSM Masking
CHES CTF 2018 [22]	Power	ARM Cortex-M4 (32 bits)	168MHz	Boolean Masking
ESHARD-AES128 [39]	EM	ARM Cortex-M4 (32 bits)	30MHz	Boolean Masking

The side-channel leakages of four of them, namely ASCADr, ASCADf, DPAv4.2, and CHES CTF 2018, are the same as adopted for the NOPOI scenario in [34] (see Sect. 2.3 and Table reftab:hpspssearch of [34] for specific details of the selected intervals). The raw side-channel measurements from ASCADr, ASCADf, DPAv4.2, and CHES CTF 2018 contain large traces with 100 000, 250 000, 150 000, and 150 000 sample points per trace, respectively. Working with such large intervals is computationally intensive, and in this paper, we also consider window resampling with a window of 20 and a step of 10. The resampled datasets result in preprocessed side-channel measurements with 25 000, 10 000, 15 000, and 15 000 samples per trace, and we consider 200 000, 50 000, 70 000, and 30 000 measurements as profiling sets for ASCADr, ASCADf, DPAv4.2, and CHES CTF 2018, respectively. For all datasets, we consider 5 000 measurements as validation sets and another 5 000 as attack sets.

The fifth dataset is ESHARD-AES128, and it consists of side-channel measurements collected from a software-masked AES-128 implementation running on an ARM Cortex-M4 device. The AES implementation is protected with a first-order Boolean masking scheme and shuffling of the **S**-box operations. In this work, we consider a trimmed version of the dataset that is publicly available⁴ and includes the processing of the masks and all **S**-box operations in the

⁴ https://gitlab.com/eshard/nucleo_sw_aes_masked_shuffled.

first encryption round without shuffling. This dataset contains 100 000 measurements, which are split into groups of 90 000, 5 000, and 5 000 for profiling, validation, and attack sets, respectively.

ASCADf and ESHARD-AES128 datasets are the only datasets in which the profiling, validation, and attack keys are equal and fixed. For the rest of the datasets, profiling, validation, and attack keys differ.

4.2 CGAN Hyperparameter Search

Through preliminary experiments, we have already confirmed that identifying effective generator and discriminator architectures is a cost-effective process, as most of the hyperparameter combinations we have tested yield satisfactory results, but better sets of hyperparameters can be found. For this purpose, we employ a random search approach with predefined hyperparameter ranges, as outlined in Table 2. Dense layers may have different numbers of neurons for the generator, and the subsequent layer never has more neurons than the previous layer. This design choice reduces the search space. Due to the limited options available for each specific hyperparameter, the number of potential generator architectures is capped at 744, while the number of potential discriminator architectures is limited to 324. Consequently, there exists a total of 241 056 possible CGAN hyperparameter selections. In addition, the generator and discriminator employ the Adam optimizer with fixed learning rates. For the discriminator, we set the learning rate to 0.0025, while for the generator, the learning rate is set to 0.0002. The discrepancy in learning rates follows [21]. Like other neural network training procedures, the CGAN training process is conducted in batches, with a fixed batch size of 400 measurements across all hyperparameter configurations and experiments from this paper. Although the batch size and learning rates could also be included in the hyperparameter search process, we decided to fix these as using poor learning-rate/batch-size combinations would result in training an unnecessarily large amount of non-converging models.

Table 2. Hyperparameter search ranges for generator and discriminator architectures.

Generator		Discriminator	
Hyperparameter	Options	Hyperparameter	Options
Dense layers	1, 2, 3, 4	Dense layers Embedding	1, 2, 3
Neurons	100, 200, 300, 400, 500	Neurons Embedding	100, 200, 500
Activation Function	linear, relu, selu, elu, leakyrelu, tanh	Dense layers Dropout	1, 2, 3
		Neurons Dropout	100, 200, 500
		Dropout Rate	0.5, 0.6, 0.7, 0.8
		Activation Function	leakyrelu

To identify the best hyperparameter setup, we need an evaluation metric. As conventional machine learning metrics are generally not suitable for assessing

models in SCA [35], we perform a profiling attack (on the validation set) on extracted features to evaluate the trained models. The target dataset X_{prof} is split into profiling and validation sets. As illustrated in Fig. 2, the input to the generator is the profiling set X_{prof} . After the CGAN training is finished for *every hyperparameter search attempt*, we predict on profiling and validation sets from X_{prof} with the trained generator. This gives us f_{prof} and f_{val} , respectively. For both sets, the keys are assumed to be known, allowing us to validate the whole process. To check how well the trained generator can extract leaky features from X_{prof} , we perform a profiling attack by training a profiling model with f_{prof} and by computing guessing entropy from f_{val} . The profiling model consists of a 4-layer MLP (each layer with 100 neurons and elu activation function) trained for 100 epochs. These hyperparameters were defined based on preliminary experiments and delivered relatively efficient profiling attack results. Here, we could also tune the profiling model architecture to find the optimal solution, which is a process that we cover in Sect. 7. The trained generator that extracts f_{prof} and f_{val} resulting in the most successful profiling attack (i.e., the profiling model that requires the least number of validation traces f_{val} to reach guessing entropy equal to one) is considered the optimal solution.

The inputs to the discriminator in the CGAN architecture include the extracted features (f_{ref} or f_{prof}) and their corresponding labels (y_{ref} or y_{prof}). The labels y_{ref} or y_{prof} refer to one output byte from the first **S-box** in the first AES encryption round: **S-box**($d_{i,j} \oplus k_{i,j}$). $d_{i,j}$ (resp. $k_{i,j}$) denote the j -th plaintext byte (resp. j -th key byte) from the i -th side-channel measurement. Only when ESHARD-AES128 is involved, the datasets are labeled according to the Hamming weight of **S-box** output bytes, i.e., $HW(\mathbf{S}\text{-box}(d_{i,j} \oplus k_{i,j}))$, as this dataset leaks in this leakage model and no successful attack results were found otherwise. Note that y_{ref} or y_{prof} need to be labeled with the same leakage model.

Next, we provide results for ASCADr reference datasets. This dataset was selected as a reference here as it provides the best results across the board. Results with different reference datasets can be found in [25, Appendix A].

4.3 ASCADr as the Reference Dataset

In our first analysis, ASCADr is considered the reference dataset. We deploy a random hyperparameter search process for each target dataset with 100 search attempts. The CGAN is trained for 200 epochs for each of these search attempts. At the end of each training epoch, we compute SNR between generated features f_{target} and secret shares, specifically, the masks and masked **S-box** output, available with the target dataset. It is important to note that, as mentioned in Sect. 3.4, these secret shares are assumed to be unknown to the attacker. However, in this context, we utilize their knowledge to provide evidence of our results.

Table 3 lists the best-found CGAN hyperparameters when ASCADf, DPAV4.2, ESHARD-AES128, and CHES CTF 2018 are considered as target datasets. Each profiling attack conducted after each hyperparameter search

attempt is applied only to the target key byte. When the target dataset is ASCADf, the target key byte is k_2 , the first masked key byte in this dataset. For the DPAv4.2, ESHARD-AES128, and CHES CTF 2018 datasets, the target key byte is k_0 .

Table 3. Best CGAN hyperparameter for different target datasets when ASCADr is a reference dataset.

Hyperparameter	Generator Network			
	ASCADf	DPAv4.2	ESHARD-AES128	CHES CTF 2018
Dense layers	1	4	3	4
Neurons	300	200-200-200-100	500-500-500	100-100-100-100
Activation Function	linear	linear	leakyrelu	linear
Hyperparameter	Discriminator Network			
	ASCADf	DPAv4.2	ESHARD-AES128	CHES CTF 2018
Dense layers Embedding	2	1	1	2
Neurons Embedding	100	100	200	200
Dense layers Dropout	3	1	1	1
Neurons Dropout	200	200	200	200
Dropout Rate	0.7	0.8	0.7	0.5
Activation Function	leakyrelu	leakyrelu	leakyrelu	leakyrelu

After finding the best CGAN architecture for each target dataset when ASCADr is set as the reference dataset, we repeat the CGAN training plus the profiling attack for the rest of the key bytes in the target dataset.

Our datasets are all first-order masked AES implementations. The extracted features f_{target} should contain leakages from the masked **S-box** output byte and the mask. However, as the Boolean masking operation is commutative, the generator cannot know what secret share should be the first or the second share. However, the order of secret shares in the generator’s output has no impact on the whole process as long as the generator can extract leaky features from the two secret shares from X_{target} . Notably, for masking schemes where the order of share values matters for recombination, the generator should learn to order shares accordingly.

Figure 4 shows the evolution of the maximum SNR values for each secret share during CGAN training. This plot illustrates the results for all target key bytes, and the average SNR is illustrated in blue for share 1 and orange for share 2. The results are provided for ASCADf, DPAv4.2, and ESHARD-AES128 as target datasets.⁵ Note that these figures also show the maximum SNR values

⁵ As masks are unavailable for the CHES_CTF 2018 dataset, we cannot perform this analysis for this target.

from the f_{ref} (in dashed green line) and X_{target} (dashed red line), which are averaged over SNR obtained from secret shares associated to each key byte. As we can see, for all target key bytes, the generator can extract features f_{target} from X_{target} , which results in high SNR values. This confirms that our proposed CGAN structure can efficiently extract features from high-order leaky points. In Sect. 6, a visualization analysis is applied to the generator to express in more detail what features are extracted from X_{target} .

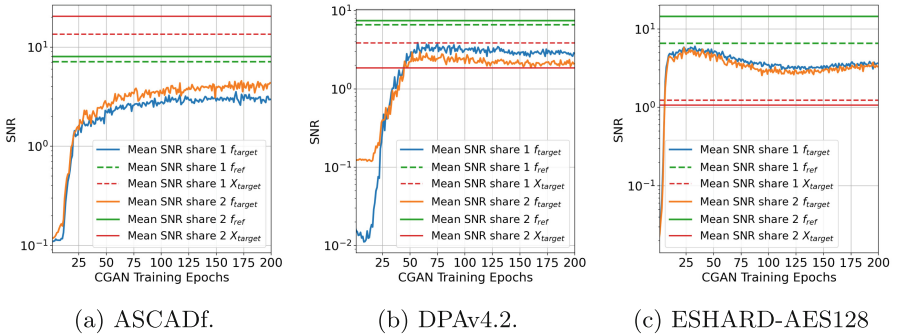


Fig. 4. Performance of CGAN architecture against different target datasets, X_{target} , when ASCADr is the reference dataset.

Another interesting outcome from the results shown in Fig. 4 is that when DPAv4.2 and ESHARD-128 are the target datasets, the averaged SNR levels from f_{target} (extracted features) are higher than the SNR levels from X_{target} (raw datasets). This occurs due to averaging but also because for some of the target key bytes, the SNR from f_{target} is higher than the SNR from X_{target} . This emphasizes the high capability of the trained generator to act as an efficient feature extractor.

4.4 Profiling Complexity of the CGAN-SCA Framework

In this section, we evaluate the profiling complexity of the CGAN as a feature extractor and its impact on the complexity of a black-box profiling attack on the target dataset. This is conducted by varying the number of measurements from X_{prof} that is considered for the training of generator and discriminator architectures. This will allow us to check whether using reduced X_{prof} datasets still provides efficient applications of our proposed CGAN-SCA framework. An attacker that is limited in the number of measurements from the target device is a realistic assumption, and having a framework that works well under these circumstances is beneficial for security assessments. Note that we do not limit the number of traces that can be collected from a reference implementation as in our extended threat model. We assume this is not a serious limitation for an attacker.

In this experimental setup, we first find hyperparameters that work in more limited scenarios. The best architectures found in Sect. 4.2 using the full target datasets do not generalize well when fewer traces are available. Thus, we repeat

the random hyperparameter search using the ranges provided in Sect. 4.2. We run 100 search attempts combination and select the best generator and discriminator architectures using the validation metric explained in Sect. 3.4. For this random search, we considered 30 000 traces from X_{prof} .

Using the best-found generator and discriminator architectures, we then train CGAN models using the reference dataset and 10 000 through 70 000 traces with 10 000 trace steps. To check how the CGAN model trained with different numbers of profiling traces X_{prof} impacts the performance of a black-box profiling attack, we ran a random search using the obtained f_{prof} with varying numbers of profiling traces. As the main idea here is to focus on the process that is efficient with less hyperparameter tuning efforts with respect to finding a good profiling model, we decided to limit the profiling model size to small MLP networks with up to four hidden layers. The hyperparameter ranges for the profiling attack model search are shown in Table 4.

Table 4. Hyperparameter search ranges for MLP as a profiling attack model.

Hyperparameter	Options
Dense layers	1, 2, 3, 4
Neurons	20, 50, 100, 200, 300, 400, 500
Activation Function	elu, selu, relu, leakyrelu, tanh
Learning Rate	0.001, 0.005, 0.0001, 0.0005
Batch Size	100, 200, 300, 400
Weight Initialization	random (normal/uniform), he (normal/uniform), glorot (normal/uniform)

For comparison, we also run these attacks in a white-box (WB) profiling scenario (following the white-box DL setup in Sect. 7) where features from X_{prof}/X_{target} are selected based on SNR.

As can be seen in Table 5, the CGAN framework can be used even in scenarios where only limited profiling traces are available from the target device. The columns indicate the number of profiling traces X_{prof} considered for training the CGAN architecture. Successful attacks are possible with only 10 000 profiling traces in both tested scenarios. While the attack results are not as efficient as with more traces, the ability of the CGAN network to learn in this limited scenario is somewhat surprising as the conventional discriminative DLSCA models often require significantly more profiling traces to generate efficient models [30]. In fact, our results are more aligned with the scheme-aware adversary who utilizes knowledge of the masking scheme to explicitly embed the combination of secret shares into a neural network layer (namely, the Grouprecombine- [30] and Bilinear [9] layers). As such, we note that including reference traces has similar benefits to these layers in terms of aiding the networks in learning the secret-share recombination.

Furthermore, in Table 5, we see that training a CGAN model with larger numbers of profiling traces can alleviate the need for using the full profiling set

in the subsequent attack phase. The CGAN feature selection has a similar effect to selecting features in a white-box setting. While using more profiling traces has clear benefits regarding attack performance, the feature selection provided by the CGAN makes it significantly easier for attack models to converge in limited scenarios by eliminating the presence of uninformative samples. This emphasizes that the CGAN framework can, to an extent, emulate feature selection effectively without having access to the mask shares of a target device.

Table 5. Number of traces to reach $GE = 1$ for varying numbers of profiling traces for ASCADr vs. DPAv4.2

Profiling Traces	CGAN training traces							
	10 000	20 000	30 000	40 000	50 000	60 000	70 000	WB
70 000	-	-	-	-	-	-	9	1
67 500	-	-	-	-	-	-	11	1
65 000	-	-	-	-	-	-	10	1
62 500	-	-	-	-	-	-	9	1
60 000	-	-	-	-	-	19	10	2
57 500	-	-	-	-	-	21	10	1
55 000	-	-	-	-	-	20	10	1
52 500	-	-	-	-	-	25	11	2
50 000	-	-	-	-	21	20	11	1
47 500	-	-	-	-	22	25	12	1
45 000	-	-	-	-	22	26	10	2
42 500	-	-	-	-	24	22	11	2
40 000	-	-	-	29	25	25	11	2
37 500	-	-	-	35	27	29	11	2
35 000	-	-	-	31	25	22	11	2
32 500	-	-	-	35	26	23	15	2
30 000	-	-	14	36	25	29	12	2
27 500	-	-	11	35	28	30	15	2
25 000	-	-	15	37	29	29	15	2
22 500	-	-	18	38	27	28	20	2
20 000	-	28	21	39	33	28	19	3
17 500	-	30	21	45	31	34	27	3
15 000	-	37	26	58	45	56	34	3
12 500	-	32	33	72	70	66	60	4
10 000	627	48	38	99	89	155	96	4
7 500	680	56	46	131	143	107	203	5
5 000	797	91	78	251	252	372	285	12

5 The Analysis of the Latent Space

In this section, we analyze how variations in the construction of f_{ref} can impact how the generator network performs at extracting features. We first look at the effect of organizing leaky features in f_{ref} in various ways and whether the generator network can mimic these patterns accurately. Second, we investigate whether f_{ref} can also be created using alternative pre-processing methods, such as PCA and LDA.

5.1 Varying f_{ref} Leakage Pattern

Here, we analyze whether the generator network in the CGAN framework can mimic the leakage patterns present in the adversarial set f_{ref} . This analysis provides more insights into the relationship between the generator and discriminator. As explained before, the generator needs to extract main features from X_{target} , and it is important to confirm if these extracted features f_{target} follow the pattern from reference features f_{ref} . This is an expected outcome from the generator as it follows the principle of GAN architectures where the generator is trained to produce outputs that are statistically similar to the adversarial dataset (which, in our case, is given by f_{ref}).

This analysis considers ASCADr to be the reference dataset and ASCADf to be the target dataset. This scenario was chosen as these datasets have very high SNR peaks concerning their secret shares and are of the same implementation and device model, simplifying the analysis without expensive hyperparameter tuning efforts. Note, however, that these datasets were acquired with distinct acquisition settings.

From the SNR-based feature selection process on ASCADr, we select 50 features for each secret share to have a total of $N_f = 100$. Thus, we organize these features in two different patterns, as shown in Figs. 5a and 5c. During the training of the CGAN architecture, at the end of each epoch, we compute the SNR levels for the secret shares on f_{target} , provided by the generator. Note in the results given in Figs. 5b and 5d how the generator learns to mimic precisely the leakage distributions from f_{ref} . These plots represent the range of minimum and maximum SNR values obtained during CGAN training epochs (i.e., we compute the SNR at the end of each CGAN training epoch). The solid lines represent the mean SNR values. These results confirm that our generator can extract leaky features from the input target traces X_{target} . An essential insight derived from this analysis is the significant role played by the feature selection process in transforming X_{ref} into f_{ref} for the generator’s feature extraction task. The number and distribution of leaky points of interest in f_{ref} directly impact the generator’s performance on its task.

5.2 Varying Reference Feature Selection Method

While it is clear that the generator can effectively emulate feature selection of SNR peaks, this method is relatively straightforward when compared with

methods currently used in literature, like LDA [6] or PCA. It is interesting to verify whether our framework allows alternative feature selection methods to be emulated. To this end, we run experiments using LDA and PCA for constructing f_{ref} . For both methods, we first select the 100 highest SNR features for each share and then transform these features into 5 components per share. Thus, in total, the number of features becomes $N_f = 10$. To test whether the framework can also emulate these methods, we run attacks against DPAv4.2 using ASCADr as a reference. To tune models for these cases, we run a hyperparameter search using the same ranges as in Sect. 4.2.

In Fig. 6, we see that the more complex feature selection methods used for the reference dataset still result in converging generators. After training generator and discriminator models, when both PCA and LDA are taken into account for feature selection from the reference dataset, we apply profiling attacks on extracted features f_{prof} and f_{target} . We can retrieve the correct key byte with 4 and 3 traces for PCA and LDA, respectively. The final performance is similar to the performance of the generators in Sect. 4, and the attack performance is comparable to the attacks with the same datasets in Table 6. From these results, we can conclude that the CGAN framework is not limited to only using SNR-based feature selection and also performs well for alternative solutions.

6 Visualizing Generator’s Feature Extraction with LRP Attribution Method

In the previous section, we demonstrated that the generator effectively extracts features from X_{target} by mimicking the pattern observed in f_{ref} . Additionally, we also verified that the feature selection method to produce f_{ref} has little impact on the whole CGAN-SCA results. This section applies the Layer-wise Relevance Propagation (LRP) [2] method to analyze the generator further. LRP is a cost-effective solution that provides interpretability and, for our case, confirms that the generator accurately captures leakage from actual leaky points of interest from X_{target} . The primary objective of this section is to present evidence that the generator, although not conditioned with labels, can extract features from the high-order leaky points of interest rather than functioning solely as a preprocessing step that leads to dimensionality reduction.

In Fig. 7, we provide two scenarios. The figure on the top-left shows the LRP values obtained from the trained generator when the reference dataset is ASCADr, and the target dataset is ASCADf. The generator’s output produces f_{target} with $N_f = 100$ features per trace. For this case, the selected pattern for f_{ref} is exactly what is shown in Fig. 5a. Thus, as the generated features f_{target} have the same shape as shown in Fig. 5b, we compute LRP for the first 50 features for share 1 and the other 50 features for share 2. Comparing with the SNR values obtained from the same target key byte of ASCADf (plot on the bottom-left of Fig. 7), we see that the generator extracts the correct features from X_{target} .

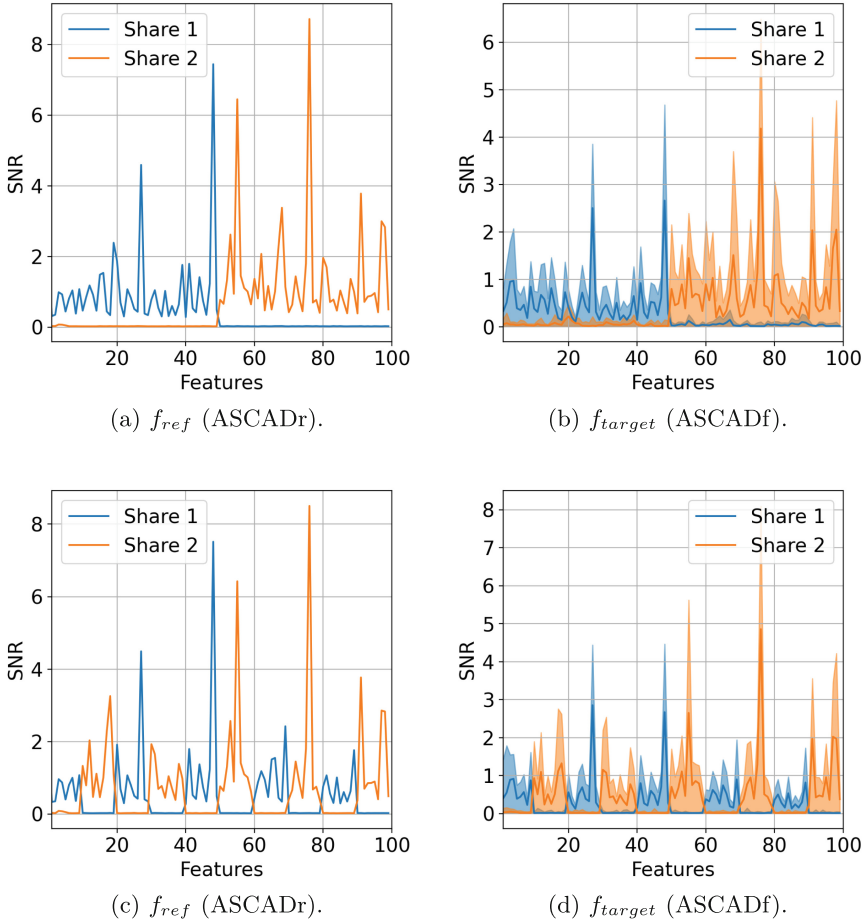
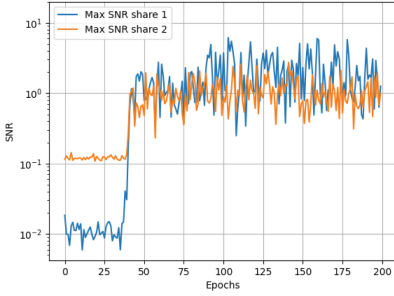
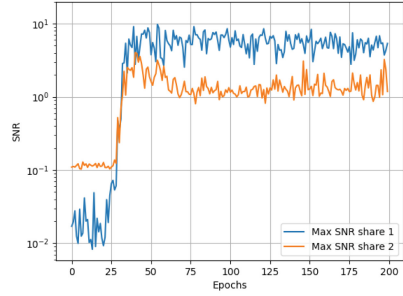


Fig. 5. SNRs of f_{ref} (left) with the corresponding f_{target} (right).

Furthermore, we present an example using the ESHARD-128 dataset. In this case, the generator is trained with ASCADf as the reference dataset. Following the same process as in the previous example, we obtain the results depicted on the right side of Fig. 7. It is noteworthy how the generator can extract features that align with the location of SNR peaks concerning the processing of high-order leakages. This interpretability analysis confirms the generator’s effectiveness in extracting high-order leakages from a target dataset when it is not even conditioned to any label class. Indeed, only conditioning the discriminator in our proposed CGAN structure is enough to implement efficient feature extraction from masked datasets. However, as the CGAN structure never sees the labels from the target attack set and is still able to extract features from this



(a) PCA



(b) LDA

Fig. 6. Maximum SNR evolution for the best model hyperparameter search ASCADr vs. DPAv4.2 using LDA/PCA for generating f_{ref} .

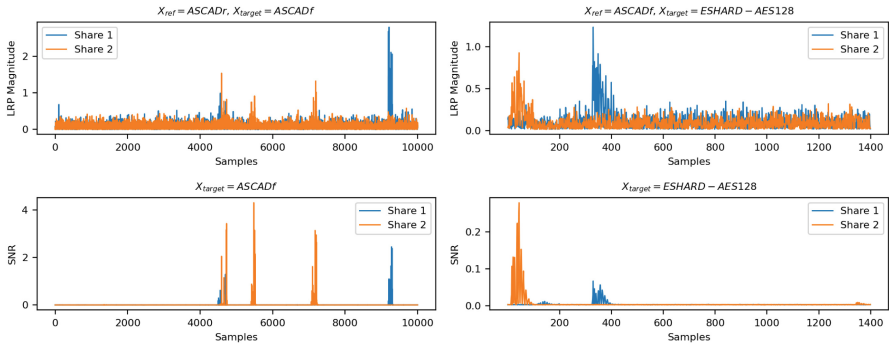


Fig. 7. Comparison between LRP magnitude and SNR values from secret shares obtained for a single target key byte.

attack set efficiently, we may intuitively conclude that the generator learns to extract input features from specific positions. The results in this section provide conditions to make the application of the CGAN-SCA framework to black-box profiling attacks more interpretable. It points out the locations in the target dataset X_{target} , where feature extraction can expose potential vulnerabilities in the implemented countermeasures.

7 Profiling Attacks and Comparison with State-of-the-Art

We employ state-of-the-art profiling attack methods as a benchmark to compare against our results. More precisely, we compare the number of attack traces that are necessary to achieve guessing entropy equal to 1 when the attack considers up to 2000 traces. Moreover, we compare the success of a hyperparameter search process. The following analysis is conducted for each dataset:

- **CGAN-SCA with DL-based profiling attack (CGAN-SCA)**: this attack is implemented with the CGAN-SCA framework presented in Sect. 3.2. The CGAN-SCA architecture is trained to achieve an efficient generator model that converts X_{prof} and X_{target} traces into f_{prof} and f_{target} . After obtaining these extracted features, we apply a DL-based profiling attack.

- **DL-based black-box profiling attack (BBDL)**: in this case, we apply DL-based profiling SCA on datasets without feature selection. The attack is considered a black box as the profiling phase does not consider any knowledge about countermeasures or secret randomness.

- **DL-based white-box profiling attack (WBDL)**: this profiling attack assumes that during profiling, an adversary can implement feature selection as countermeasures (i.e., the masking scheme) and secret randomness (i.e., secret masks) are known. Therefore, feature or points of interest selection can be applied to profiling and attack traces.

- **White-box Gaussian Template Attack (WBTA)**: this process follows a white-box profiling attack in which points of interest are selected based on the set of highest SNR peaks obtained with the knowledge of secret masks. For all scenarios, we select 1000 points of interest by targeting a second-order leakage function (500 points of interest for each share), which is reduced with linear discriminant analysis (LDA) to 10 points of interest. Afterward, we build Gaussian templates with them.

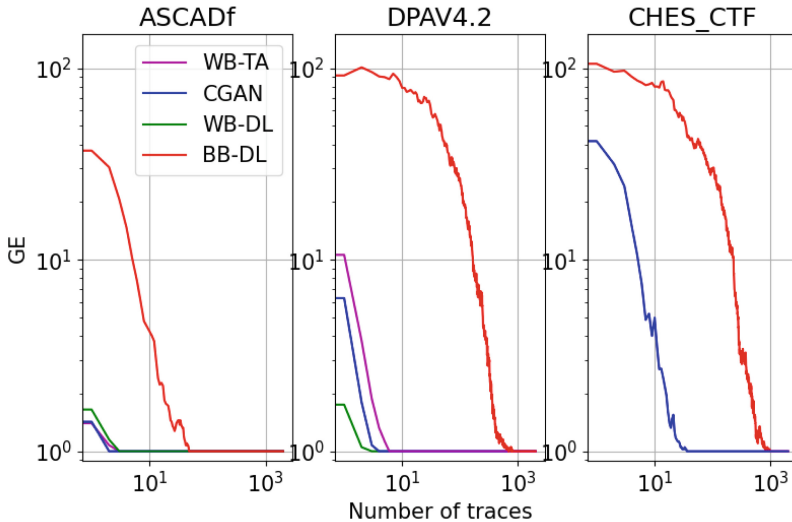


Fig. 8. GE results for key-byte 2 for various targets and methods (ref: ASCADr)

The first three profiling methods, which consist of deep learning-based profiling models, include a hyperparameter tuning process for a small MLP model.

For each of the 16 target key bytes from the full AES 128-bit key, we search for 100 random MLP architectures using the same hyperparameter ranges from Table 4. Each of these MLP architectures is then trained, validated, and tested separately with:

1. f_{prof} , f_{target} , and f_{val} sets, respectively, obtained by predicting the generator \mathcal{G} with the profiling, validation, and attack sets from the X_{prof} and X_{target} . This way, we implement the aforementioned **CGAN-SCA with DL-based profiling attack**;
2. original X_{prof} (split into profiling and validation traces) and X_{target} , to implement the aforementioned **DL-based black-box profiling attack**: BBDL.
3. SNR-based selected features from X_{prof} and X_{target} to implement the aforementioned **DL-based white-box profiling attack**: WBDL.

Through this comparison, we emphasize the significantly reduced effort from the CGAN-SCA approach in finding an efficient profiling model that shows performance comparable to optimal profiling models, as is expected for WBDL and WBTA. Table 6 provides the performance of the five aforementioned profiling attack methods on datasets listed in Sect. 4.1. For the case of CGAN-SCA methods, we provide results for different reference datasets. This table shows results with different colors to differentiate among profiling attack categories for better readability.

As can be seen in Fig. 8 and Table 6, the attacks using ASCADr as a reference for all targets improve substantially over the BBDL attacks. Furthermore, in the best-case scenarios for ASCAD(r/f) and DPAv4.2, results are comparable to attacks following white-box assumptions.⁶ Only for ESHARD, and when DPAv4.2 is used as a reference, we see that white-box attacks still substantially outperform our attacks. We mostly attribute this to the larger difference in implementations/devices, which we discuss in more depth in Sect. 8.

Table 7 shows the search success from the hyperparameter search part of DL-based profiling attack methods. The search success indicates the percentage of times a profiling model has reached the guessing entropy of 1 with less than 2 000 attack traces. The percentages are the average of all target key bytes. CGAN-SCA and WBDL present similar performances and are significantly superior to black-box DL. This finding is impressive if we remember that CGAN-SCA is a black-box (i.e., non-worst case) profiling approach. The results from Table 7 corroborate what was already shown in [34]: spending significant effort on hyperparameter search process eventually results in a high-performing deep neural network against first-order masking in AES implementations. However, what matters in this table is the search success, which informs more about the chances of finding a good group of hyperparameters and training settings. Although black-box DL-based profiling attacks result in successful attacks with (in some cases) very few required attack traces, the search success with CGAN-SCA framework and white-box DL approaches are significantly higher. For instance, when

⁶ While it seems likely that CHES_CTF 2018 results are competitive with white-box attacks, we cannot verify this as mask values are not available.

Table 6. The minimum number of attack traces to obtain guessing entropy equal to 1. The symbol **x** indicates that the target key byte is not recovered with 2000 attack traces. The **NA** indicates that the attack is not applicable because the target key bytes are unprotected.

Target	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
	ASCADr															
CGAN-SCA (ref: ASCADr)	NA	NA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
CGAN-SCA (ref: DPAv4.2)	NA	NA	4	2	2	2	5	11	10	2	6	8	5	2	2	2
White-box DL	NA	NA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Black-box DL	NA	NA	1	2	3	1	29	5	1	16	9	9	9	6	1	1
White-box TA	NA	NA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	ASCADf															
CGAN-SCA (ref: ASCADr)	NA	NA	1	1	1	1	1	1	4	1	5	1	15	5	1	5
CGAN-SCA (ref: DPAv4.2)	NA	NA	3	3	2	2	2	3	7	2	5	2	7	5	2	5
White-box DL	NA	NA	1	1	1	1	1	1	4	1	2	1	3	3	1	1
Black-box DL	NA	NA	9	6	9	8	18	x	27	6	20	6	x	34	17	7
White-box TA	NA	NA	1	1	1	1	1	1	4	1	2	1	8	4	1	5
	DPAv4.2															
CGAN-SCA (ref: ASCADr)	1	3	2	5	3	2	4	2	7	2	3	3	2	2	5	2
CGAN-SCA (ref: ASCADf)	5	2	3	12	3	6	5	9	4	3	7	6	3	2	5	8
White-box DL	1	1	1	2	3	1	2	1	3	1	2	2	1	1	2	1
Black-box DL	x	315	140	x	x	x	1454	x	x	x	x	x	x	x	x	x
White-box TA	3	2	2	3	2	3	3	2	5	3	4	3	2	2	3	2
	CHES CTF 2018															
CGAN-SCA (ref: ASCADr)	36	24	22	20	51	19	21	36	34	18	25	23	24	22	22	19
CGAN-SCA (ref: ASCADf)	19	39	27	30	19	14	22	25	32	15	29	30	33	18	27	18
CGAN-SCA (ref: DPAv4.2)	91	47	36	115	159	200	73	138	858	56	136	50	557	78	124	52
Black-box DL	x	471	367	77	668	1327	304	1216	1369	957	83	662	x	459	413	380
	ESHARD-AES128															
CGAN-SCA (ref: ASCADr)	556	1105	312	224	709	257	396	206	967	385	244	272	309	294	292	299
CGAN-SCA (ref: ASCADf)	491	1248	528	357	1539	532	641	493	622	552	373	513	732	406	454	572
CGAN-SCA (ref: DPAv4.2)	1353	x	x	1242	x	x	x	1051	x	1787	1643	x	x	x	x	x
White-box DL	640	1546	875	727	x	774	799	667	x	846	487	745	1162	649	818	1037
Black-box DL	758	748	625	616	1957	950	536	700	x	769	846	479	1527	769	572	462
White-box TA	67	81	97	89	110	75	123	107	152	100	100	89	127	87	111	111

Table 7. Search success for MLP-based profiling attack with random hyperparameter search. The percentage indicates the number of successful MLP models out of 100, and it is averaged for all target key bytes.

Method	CGAN-SCA	CGAN-SCA	CGAN-SCA	White-box	Black-box
Target	(ref: ASCADr)	(ref: ASCADf)	(ref: DPAv4.2)	DL	DL
ASCADr	NA	72.80%	90.47%	99.88%	8.92%
ASCADf	64.22%	NA	68.25%	70.58%	9.24%
DPAv4.2	65.07%	62.16%	NA	63.68%	0.74%
CHES CTF 2018	61.10%	99.55%	33.15%	NA	12.14%
ESHARD-AES128	94.56%	54.48%	10.17%	63.68%	35.60%

ASCADr is set as a reference, and DPAv4.2 is set as a target, the search success for a black-box DL is 0.74%, while for CGAN-SCA is 65.07%. For the case when ASCADf is the reference and CHES CTF 2018 is the target, the search success increases from 12.14% for a black-box approach to 99.55% with our CGAN-SCA framework. This justifies the need for feature selection (in the case of white-box) or feature extraction (in the case of CGAN-SCA framework) to speed up security evaluations. Since our proposed solution is also black-box, it becomes very attractive for efficiently assessing the security of masked implementations.

8 Discussion

Profiling attack results presented in this paper are aligned with the state-of-the-art for the evaluated datasets (see [34] for the ASCAD, CHES CTF 2018, and DPAv4.2 datasets. To the authors' knowledge, there are no published ESHARD-AES128 dataset results for profiling attacks). Such results were possible due to the following additional elements in a security assessment process:

1. **Using a (white-box) reference dataset.** The CGAN-SCA structure requires a reference device with similar implementation specifications to the target one. This paper shows that reference and target datasets can be gathered from different devices, cryptographic designs (with at least the same cryptographic algorithm and a similar masking scheme), varying source codes, and different acquisition setups. For some experimental examples, reference and target datasets come from different side-channel types (e.g., power and electromagnetic analysis). Together with the availability of a reference implementation, it should also be possible to implement feature selection from this same implementation. This paper assumes that secret masks from the reference implementation are known to compute feature selection.
2. **The employment of a generative model for feature extraction from target side-channel measurements.** As specified in Sect. 3.2, the CGAN-SCA framework can implement feature extraction from a target dataset, and a reference dataset is used as an adversarial dataset. We are aware that this whole process increases the complexity of the analysis because a CGAN architecture (i.e., generator and discriminator neural networks) needs to be trained before applying a profiling attack on the extracted features from the target dataset. However, our experimental analysis demonstrated that when an efficient CGAN architecture is found, and the extracted features contain high SNR levels concerning the leakage of intermediate variable (e.g., masks and masked S-Box outputs), defining a profiling model becomes relatively easy. Therefore, in practice, the efforts to find an efficient profiling model (see [34] where the authors performed very costly hyperparameter tuning processes) are transferred to defining an efficient CGAN architecture.
3. **Hyperparameter tuning for generator and discriminator models.** An efficient CGAN architecture requires some carefully tuned generator and discriminator models. Overall, this is the only time-consuming part of the proposed CGAN-SCA framework. However, this whole process brings clear

benefits, as a feature extraction process from raw side-channel measurements becomes possible without assuming any knowledge about low-level countermeasure details and secret randomness.

Our results present three broad categories of ‘similar’ implementations, allowing us to give some takeaways on how similar the reference implementation must be:

1. **ASCAD(f/r) vs. ASCAD(f/r)**: The reference device and implementation are the exact same. This scenario can occur when an attacker/evaluator has access to the source code of an implementation but cannot alter this implementation on the target device. In such a scenario, the attacker/evaluator could utilize an instrumented version of the source to create a reference dataset. Our results in Sect. 4 show that the inclusion of this reference implementation results in significantly improved attack results over black-box DL attacks, and the results are competitive with white-box approaches.
2. **ASCAD(f/r) vs. DPAv4.2**: The reference and target devices are similar in that both are 8-bit micro-controllers with RISC-based micro-architectures. The measurements for these targets are in different side-channel domains (EM for ASCAD vs Power for DPA). Both implementations incorporate Boolean masking-based countermeasures, although the specifics of the implementations differ somewhat. For DPAv4.2, an RSM-based masking scheme is employed, which results in 16 possible mask values, while for both ASCAD versions, we have 256 possible mask values. Results here still showcase strong improvements over black-box DL, especially when DPAv4.2 is the target, but the attacks are somewhat less efficient than white-box attacks.
3. **(ASCAD/DPA) vs. (CHES_CTF/ESHARD)**: We target 32-bit micro-controllers with ARM micro-architectures while using 8-bit AVR micro-controllers as the reference. The implementations are broadly similar in that these are all software AES implementations protected with first-order Boolean masking. As we see in Table 6, the attacks against both CHES_CTF and ESHARD are better than the black-box DL attacks when ASCAD is used as the reference, while the results are similar to (CHES_CTF), or worse than (ESHARD) black-box attacks when DPA is used as the reference. Additionally, we see that for ESHARD, the performance of white-box attacks is still significantly better than that of our CGAN-SCA setups.

In ideal cases where the reference implementation only differs in terms of allowing the knowledge of mask values,⁷ we see that results are competitive with white-box attacks. The device model and architecture similarity are more important than countermeasure implementation for other settings. While our results do not allow for strong requirements on the reference implementation, overall, the necessary ‘similarity’ to improve over black-box attacks is not extremely stringent. In some of the tested settings where the devices differ in terms of micro-architecture and implementation, we still see improvements over black-box

⁷ The ASCAD(f/r) vs. ASCAD(f/r) scenario.

attacks although the performance in these cases is worse than their white-box counterparts. In addition, it is more important to ensure the devices are similar in terms of, e.g., micro-architecture or bus size, over specific countermeasure implementation details (i.e., RSM vs. Boolean masking).

9 Conclusions and Future Work

This paper proposes a novel CGAN-based framework to automatically extract features from a target dataset when the adversarial dataset comes from a similar, open, and fully controlled implementation. Our solution differs from conventional CGAN architectures from the literature: the generator receives real (target) traces instead of noise, and it is not conditioned with label class, allowing it to extract features from an unlabeled attack set. By applying our framework to five publicly available masked AES datasets, we obtain profiling attack results that significantly surpass the state-of-the-art black-box security assessment and rival the performance of worst-case (white-box) security evaluations. The proposed CGAN-SCA framework can precisely extract features from high-order leakages by mimicking the feature distribution present in a reference dataset. Our method makes hyperparameter tuning in a deep learning-based profiling attack almost negligible, similar to white-box deep learning-based security evaluations.

For future work, we plan to investigate the effectiveness of CGAN architectures to extract features from high-order masking schemes. Moreover, we plan to implement more complex generator and discriminator models, such as CNN-based architectures, which could extract features from desynchronized datasets. More complex CGAN structures could potentially reduce some of our framework's limitations, such as using a reference dataset with a minimum acceptable SNR level regarding the n secret shares. A way to define a cost-efficient early stopping metric during CGAN training could also be an interesting research direction. Finally, we plan to explore whether the proposed structure can be adapted to non-profiling settings.

Acknowledgements. This work was performed using the ALICE compute resources provided by Leiden University.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2002*, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. *Lecture Notes in Computer Science*, vol. 2523, pp. 29–45. Springer (2002). https://doi.org/10.1007/3-540-36400-5_4
2. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE* **10**(7), 1–46 (07 2015). <https://doi.org/10.1371/journal.pone.0130140>

3. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptographic Engineering* **10**(2), 163–188 (2020). <https://doi.org/10.1007/s13389-019-00220-8>
4. Bhasin, S., Bruneau, N., Danger, J., Guilley, S., Najm, Z.: Analysis and improvements of the DPA contest v4 implementation. In: Chakraborty, R.S., Matyas, V., Schaumont, P. (eds.) *Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings. Lecture Notes in Computer Science*, vol. 8804, pp. 201–218. Springer (2014). https://doi.org/10.1007/978-3-319-12060-7_14, https://doi.org/10.1007/978-3-319-12060-7_14
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. Lecture Notes in Computer Science*, vol. 3156, pp. 16–29. Springer (2004). https://doi.org/10.1007/978-3-540-28632-5_2
6. Bronchain, O., Cassiers, G., Standaert, F.: Give me 5 minutes: Attacking ASCAD with a single side-channel trace. *IACR Cryptol. ePrint Arch.* p. 817 (2021), <https://eprint.iacr.org/2021/817>
7. Bronchain, O., Durvaux, F., Masure, L., Standaert, F.: Efficient profiled side-channel analysis of masked implementations, extended. *IEEE Trans. Inf. Forensics Secur.* **17**, 574–584 (2022). <https://doi.org/10.1109/TIFS.2022.3144871>
8. Bronchain, O., Standaert, F.: Side-channel countermeasures’ dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**(2), 1–25 (2020). <https://doi.org/10.13154/tches.v2020.i2.1-25>
9. Cao, P., Zhang, C., Lu, X., Gu, D., Xu, S.: Improving deep learning based second-order side-channel analysis with bilinear CNN. *IEEE Trans. Inf. Forensics Secur.* **17**, 3863–3876 (2022). <https://doi.org/10.1109/TIFS.2022.3216959>
10. Cao, P., Zhang, H., Gu, D., Lu, Y., Yuan, Y.: AL-PA: cross-device profiled side-channel attack using adversarial learning. In: Oshana, R. (ed.) *DAC ’22: 59th ACM/IEEE Design Automation Conference, San Francisco, California, USA, July 10 - 14, 2022.* pp. 691–696. ACM (2022). <https://doi.org/10.1145/3489517.3530517>
11. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science*, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3
12. Chen, X., Duan, Y., Houthoofd, R., Schulman, J., Sutskever, I., Abbeel, P.: Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) *Advances in Neural Information Systems Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain.* pp. 2172–2180 (2016), <https://proceedings.neurips.cc/paper/2016/hash/7c9d0b1f96aebd7b5eca8c3edaa19ebb-Abstract.html>
13. Cristiani, V., Lecomte, M., Maurine, P.: The evil machine: Encode, visualize and interpret the leakage. In: *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing.* p. 1566-1575. SAC ’23, Association for Computing Machinery, New York, NY, USA (2023). <https://doi.org/10.1145/3555776.3577688>
14. Dubrova, E., Ngo, K., Gärtner, J.: Breaking a fifth-order masked implementation of crystals-kyber by copy-paste. *IACR Cryptol. ePrint Arch.* p. 1713 (2022), <https://eprint.iacr.org/2022/1713>

15. Genevey-Metat, C., Gérard, B., Heuser, A.: On what to learn: Train or adapt a deeply learned profile? IACR Cryptol. ePrint Arch. p. 952 (2020), <https://eprint.iacr.org/2020/952>
16. Genevey-Metat, C., Heuser, A., Gérard, B.: Trace-to-trace translation for SCA. In: Grosso, V., Pöppelmann, T. (eds.) Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13173, pp. 24–43. Springer (2021). https://doi.org/10.1007/978-3-030-97348-3_2
17. Genkin, D., Shamir, A., Tromer, E.: RSA key extraction via low-bandwidth acoustic cryptanalysis. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 444–461. Springer (2014). https://doi.org/10.1007/978-3-662-44371-2_25
18. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5154, pp. 426–442. Springer (2008). https://doi.org/10.1007/978-3-540-85053-3_27
19. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), <http://www.deeplearningbook.org>
20. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. pp. 2672–2680 (2014), <https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html>
21. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 6626–6637 (2017), <https://proceedings.neurips.cc/paper/2017/hash/8ald694707eb0fef65871369074926d-Abstract.html>
22. Hu, Y., Zheng, Y., Feng, P., Liu, L., Zhang, C., Gohr, A., Jacob, S., Schindler, W., Buhan, I., Tobich, K.: Machine learning and side channel analysis in a CTF competition. IACR Cryptol. ePrint Arch. p. 860 (2019), <https://eprint.iacr.org/2019/860>
23. Hutter, M., Schmidt, J.: The temperature side channel and heating fault attacks. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8419, pp. 219–235. Springer (2013). https://doi.org/10.1007/978-3-319-08302-5_15
24. Isola, P., Zhu, J., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017. pp. 5967–5976. IEEE Computer Society (2017). <https://doi.org/10.1109/CVPR.2017.632>

25. Karayalçın, S., Krcek, M., Wu, L., Picek, S., Perin, G.: It's a kind of magic: A novel conditional GAN framework for efficient profiling side-channel analysis (extended version). Cryptology ePrint Archive, Paper 2023/1108 (2023), <https://eprint.iacr.org/2023/1108>
26. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer-Verlag (1996)
27. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25
28. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. pp. 388–397. CRYPTO '99, Springer-Verlag, London, UK, UK (1999), <http://dl.acm.org/citation.cfm?id=646764.703989>
29. Kong, J., Kim, J., Bae, J.: Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), <https://proceedings.neurips.cc/paper/2020/hash/c5d736809766d46260d816d8dbc9eb44-Abstract.html>
30. Masure, L., Cristiani, V., Lecomte, M., Standaert, F.: Don't learn what you already know scheme-aware modeling for profiling side-channel analysis against masking. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 32–59 (2023). <https://doi.org/10.46586/tches.v2023.i1.32-59>
31. Masure, L., Strullu, R.: Side channel analysis against the anssi's protected AES implementation on ARM. IACR Cryptol. ePrint Arch. p. 592 (2021), <https://eprint.iacr.org/2021/592>
32. Mirza, M., Osindero, S.: Conditional generative adversarial nets. CoRR **abs/1411.1784** (2014), <http://arxiv.org/abs/1411.1784>
33. Mukhtar, N., Batina, L., Picek, S., Kong, Y.: Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. In: Galbraith, S.D. (ed.) Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13161, pp. 297–321. Springer (2022). https://doi.org/10.1007/978-3-030-95312-6_13
34. Perin, G., Wu, L., Picek, S.: Exploring feature selection scenarios for deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 828–861 (Aug 2022). <https://doi.org/10.46586/tches.v2022.i4.828-861>, <https://tches.iacr.org/index.php/TCHES/article/view/9842>
35. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 209–237 (Nov 2018). <https://doi.org/10.13154/tches.v2019.i1.209-237>, <https://tches.iacr.org/index.php/TCHES/article/view/7339>
36. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: Sok: Deep learning-based physical side-channel analysis. ACM Comput. Surv. (oct 2022). <https://doi.org/10.1145/3569577>, just Accepted

37. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) *Advances in Cryptology - EUROCRYPT 2009*. pp. 443–461. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
38. Thapar, D., Alam, M., Mukhopadhyay, D.: Deep learning assisted cross-family profiled side-channel attacks using transfer learning. In: *22nd International Symposium on Quality Electronic Design, ISQED 2021, Santa Clara, CA, USA, April 7-9, 2021*. pp. 178–185. IEEE (2021). <https://doi.org/10.1109/ISQED51717.2021.9424254>
39. Vasselle, A., Thiebeauld, H., Maurine, P.: Spatial dependency analysis to extract information from side-channel mixtures: extended version. *J. Cryptogr. Eng.* **13**(4), 409–425 (2023). <https://doi.org/10.1007/S13389-022-00307-9>
40. Veyrat-Charvillon, N., Gérard, B., Standaert, F.X.: Soft analytical side-channel attacks. In: *Advances in Cryptology—ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20*. pp. 282–296. Springer (2014)
41. Wang, P., Chen, P., Luo, Z., Dong, G., Zheng, M., Yu, N., Hu, H.: Enhancing the performance of practical profiling side-channel attacks using conditional generative adversarial networks. *CoRR abs/2007.05285* (2020), <https://arxiv.org/abs/2007.05285>
42. Wu, L., Perin, G., Picek, S.: Not so difficult in the end: Breaking the lookup table-based affine masking scheme. In: Carlet, C., Mandal, K., Rijmen, V. (eds.) *Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers. Lecture Notes in Computer Science*, vol. 14201, pp. 82–96. Springer (2023). https://doi.org/10.1007/978-3-031-53368-6_5
43. Yu, H., Shan, H., Panoff, M., Jin, Y.: Cross-device profiled side-channel attacks using meta-transfer learning. In: *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*. pp. 703–708. IEEE (2021). <https://doi.org/10.1109/DAC18074.2021.9586100>
44. Zaid, G., Bossuet, L., Carbone, M., Habrard, A., Venelli, A.: Conditional variational autoencoder based on stochastic attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2023**(2), 310–357 (2023). <https://doi.org/10.46586/tches.v2023.i2.310-357>
45. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2020**(1), 1–36 (Nov 2019). <https://doi.org/10.13154/tches.v2020.i1.1-36>, <https://tches.iacr.org/index.php/TCHES/article/view/8391>
46. Zhu, J., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. pp. 2242–2251. IEEE Computer Society (2017). <https://doi.org/10.1109/ICCV.2017.244>

ZKFault: Fault Attack Analysis on Zero-Knowledge Based Post-quantum Digital Signature Schemes

Puja Mondal¹✉, Supriya Adhikary¹, Suparna Kundu²,
and Angshuman Karmakar¹

¹ Department of Computer Science and Engineering, IIT Kanpur, Kanpur, India
{pujamondal, adhikarys, angshuman}@cse.iitk.ac.in

² COSIC, KU Leuven, Kasteelpark Arenberg 10, Bus 2452,
3001 Leuven-Heverlee, Belgium
suparna.kundu@esat.kuleuven.be

Abstract. Computationally hard problems based on coding theory, such as the syndrome decoding problem, have been used for constructing secure cryptographic schemes for a long time. Schemes based on these problems are also assumed to be secure against quantum computers. However, these schemes are often considered impractical for real-world deployment due to large key sizes and inefficient computation time. In the recent call for standardization of additional post-quantum digital signatures by the National Institute of Standards and Technology, several code-based candidates have been proposed, including LESS, CROSS, and MEDS. These schemes are designed on the relatively new zero-knowledge framework. Although several works analyze the hardness of these schemes, there is hardly any work that examines the security of these schemes in the presence of physical attacks. In this work, we analyze these signature schemes from the perspective of fault attacks. All these schemes use a similar tree-based construction to compress the signature size. We attack this component of these schemes. Therefore, our attack is applicable to all of these schemes. In this work, we first analyze the LESS signature scheme and devise our attack. Furthermore, we showed how this attack can be extended to the CROSS signature scheme. Our attacks are built on very simple fault assumptions. Our results show that we can recover the entire secret key of LESS and CROSS using as little as a single fault. Finally, we propose various countermeasures to prevent these kinds of attacks and discuss their efficiency and shortcomings.

Keywords: Post-quantum cryptography · Post-quantum signature · Code-based cryptography · Fault attacks · LESS · CROSS

1 Introduction

Digital signature schemes are one of the most used and fundamental cryptographic primitives. The security of our current prevalent digital signature

schemes based on integer factorization [33] or elliptic curve discrete logarithms [23] can be compromised using a large quantum computer [30,37]. Therefore, we need quantum computer-resistant digital signature algorithms. In 2022, the National Institute of Standards and Technology (NIST) selects three post-quantum digital signature schemes [2] CRYSTALS-DILITHIUM [15], FALCON, and SPHINCS+ [5] for standardization. Among them, FALCON and DILITHIUM are based on lattices, and SPHINCS+ is a hash-based signature scheme.

A majority of these signature schemes are lattice-based. Therefore, a breakthrough result in the field of cryptanalysis of lattice-based cryptography could create a major dilemma in the transition from classical to post-quantum cryptography. Such incidents are not very rare. Some recent examples are Castryck *et al.*'s [10] attack on the post-quantum key-exchange mechanism based on supersingular isogeny Diffie-Hellman [19] problem or Beullens's attack [6] on post-quantum digital signature scheme Rainbow [14]. Both of these schemes were finalists of the NIST's post-quantum standardization procedure. Therefore, diversification in the underlying hard problems ensures that if one of the cryptographic schemes is compromised, others may remain secure. Another problem of the currently standardized signature schemes is their very large signature sizes compared to classical signatures. This renders them almost impractical for real-world use cases like SSL/TLS certificate chains. Recognizing the critical importance of diversification and the practical use of digital signatures, NIST has recently issued an additional call [25] for post-quantum secure digital signatures. In this call, NIST emphasizes the importance of small signature and fast verification to enhance practicality.

Linear Equivalence Signature Scheme (LESS) [7,35] is a submitted digital signature scheme aimed at increasing diversification and smaller signature and public key sizes. There are other code-based submissions like WAVE [36], enhanced pqsigRM [11], and CROSS [34]. These schemes are based on the Syndrome Decoding Problem (SDP) for linear codes. The hardness of SDP relies on different variants of Information Set Decoding (ISD) algorithms. On the other hand, LESS has avoided the SDP, and it is the first cryptographic scheme based on the Code Equivalence Problem (CEP). The CEP asks to determine if two linear codes are equivalent to each other. In the Hamming metric, the notion of equivalence is linked to the existence of a monomial transformation, often termed the Linear Equivalence Problem (LEP).

Due to the choice of this hard problem, the designers could choose parameters that lead to smaller key sizes without compromising security. The designers have also proposed different compression techniques to reduce the key sizes. LESS offers a balanced trade-off between the combined public key and signature size and the efficiency of signing and verification routines. Table 4 in Appendix A compares the key sizes and efficiency of LESS and other code-based digital signature schemes.

We want to note that LESS first introduced the novel problem CEP or LEP for cryptographic constructions. It uses a 3-round interactive sigma pro-

TOCOL between a prover and a verifier. Other signature schemes like MEDS and CROSS are also based on similar zero-knowledge identification schemes. Multiple rounds of the identification scheme are used here, which is converted into a signature scheme using the Fiat-Shamir transformation. However, using multiple rounds increases the signature size. Here, we have noticed that all three signature schemes, LESS, CROSS and MEDS [12], use the same compression technique that helped the designers ease the long-enduring bottleneck of large signature sizes in code-based cryptography. However, the implementation of this common compression technique has potential vulnerabilities against fault attacks that we identified in this work. Our primary motivation in this work is to uncover potential vulnerabilities against a wide spectrum of fault attacks and propose suitable countermeasures for the schemes LESS and CROSS that use the protocols having the same compression technique. We are confident that this work will help to improve the LESS and CROSS signature schemes and be useful in the evaluation of NIST’s standardization procedure. Further, we strongly believe that this will also be beneficial to other cryptographic signature schemes, such as MEDS, as it uses a similar technique. Below, we briefly summarize our contributions.

Fault Analysis of LESS Digital Signature: We have explored several fault attack surfaces of the LESS signature scheme that could be exploited by an adversary. We found different attack surfaces in the signing algorithm of LESS, and attack strategies that can be utilized on those attack surfaces. We observed that the designers of LESS proposed a technique to compress the signature size. They used a binary tree called *Reference Tree* to fulfil this purpose. We show that the modification of the values in the tree during the signing algorithm leaks information about the secret key as part of the output signature. We further use this information to recover the full secret key.

Versatility of Our Fault Attack: Our attack assumes a single fault injection model. We want to note that our focus was to develop the theoretical framework to recover the secret after the fault injection. In this regard, our attack can be realized using many different faults. Therefore, it is very versatile *i.e.* not skewed in favour of the attacker. In particular, we discuss the applicability of our attack using different types of faults, such as instruction skip, stuck-at-zero, and bit-flip. These types of faults can be realized using different mechanisms such as voltage glitch [13], Rowhammer [24, 31], clock glitch [9, 28], laser fault injection [8], electromagnetic fault injection [17, 20] etc.

Strong Mathematical Analysis: We give detailed mathematical analysis to recover the secret key after the fault injections. We consider an arbitrary location for the fault injection, which is known to the attacker. Then, discuss the methods to recover the secret key in different scenarios. To further improve the effectiveness and practicality of our attack, we also provide a very effective method to remove noise from the experiments *i.e.* differentiating between effective and ineffective faults. This is a non-trivial problem in any fault injection attack. We mathematically derived the expected amount of secret information that can be recovered from a single effective fault.

Application to Other Zero-Knowledge Based Signature Schemes: Other code-based signature schemes in the NIST additional call for signatures such as CROSS [34] and MEDS [12], use a similar zero-knowledge framework as LESS. In these frameworks, the challenger and prover must communicate a series of challenges and responses for the soundness of schemes. This increases the signature size of the digital signature schemes designed using this framework. All these three signature schemes use a binary tree-based compression technique to reduce the signature size. As our attack targets this method, our attack strategy can also be extended to these schemes. We have explained this strategy for the CROSS signature scheme in this work.

Attack Simulation: We have an end-to-end fault attack simulation on the reference implementation of LESS and CROSS signature schemes. For LESS, we have simulated the attack in a way so that it can count the number of secret matrix recovered with one faulted signature, the number of faulted signatures required to recover the whole secret. Also, our simulation induces fault with varying successful fault probability. In both schemes, we modify a particular node of the binary tree structure and then recover the secret from the faulted signatures. We have shown that if we inject fault in a specific location, then the entire secret can be recovered from a single effective fault signature for all the parameter sets of LESS except the parameter LESS-1s. For the CROSS signature scheme, only one effective faulted signature is enough to recover the complete secret for all parameters.

Countermeasures: Finally, we discuss different countermeasures that can prevent such attacks. We show that these countermeasures are effective against the single-fault attack models. Our first countermeasure removes the primary source of vulnerability *i.e.* the generation of the *Reference Tree*. This rather simple method increases the signature size. The second countermeasure modifies the *Reference Tree* generation procedure such that the attack surfaces are eliminated. This method ensures that the signature sizes stay the same as the original LESS proposal [35]. Lastly, we implemented the second countermeasure for LESS and compared its performance with the original LESS implementation. The performance cost of our second countermeasure is the same as the cost of the original LESS implementation.

2 Preliminaries

\mathbb{Z}_q denotes the ring of integers modulo q . Additionally, \mathbb{F}_q and \mathbb{F}_q^* have been used to signify the field with q elements and the multiplicative group of this field \mathbb{F}_q , respectively. The sets \mathbb{F}_q^k and $\mathbb{F}_q^{k \times n}$ represent the collection of all vectors of size k and all matrices of dimension $k \times n$ over the field \mathbb{F}_q , respectively. We use calligraphic uppercase (\mathcal{C}) to denote a linear code.

The lowercase letters (a) and uppercase letters (A) denote the scalars and the ordered set of scalars, respectively. A^c represents the complement of the set A . We use bold lowercase (\mathbf{a}) to denote vectors in any domain, and the i -th

entry of the vector \mathbf{a} is denoted by $\mathbf{a}[i]$. We denote the i -th standard basis as \mathbf{e}_i . The transpose of a vector \mathbf{a} is denoted by \mathbf{a}^T .

The bold uppercase letters (\mathbf{A}) represent matrices. Let \mathbf{A} be a matrix, then $\mathbf{A}[i, j]$ represents the i, j -th entry of the matrix \mathbf{A} . Also, $\mathbf{A}[* , j]$ and $\mathbf{A}[i , *]$ represent the j -th column and i -th row of the matrix \mathbf{A} respectively. Let $J \subset \mathbb{Z}_n$ be an ordered set of column indices of the matrix \mathbf{A} , then the notation $\mathbf{A}[* , J]$ represents the submatrix of \mathbf{A} formed by selecting columns with indices specified in the set J . Similarly, if J is an ordered set of row indices of matrix \mathbf{A} , then the notation $\mathbf{A}[J , *]$ represents the submatrix of \mathbf{A} formed by selecting rows with indices specified in the set J . The transpose of a matrix \mathbf{A} is denoted by \mathbf{A}^T . The inner product of two vectors \mathbf{a} and \mathbf{b} of same size is denoted by $\langle \mathbf{a}, \mathbf{b} \rangle$ and is defined by $\sum_i \mathbf{a}[i]\mathbf{b}[i]$. The set of all invertible matrices of order k over \mathbb{F}_q is denoted by $GL_k(q)$.

2.1 Definitions

Definition 1. (Monomial matrix) An $n \times n$ matrix \mathbf{A} is called a monomial matrix if we can write $\mathbf{A} := (\mathbf{u}[0]\mathbf{e}_{\pi(0)} \mid \mathbf{u}[1]\mathbf{e}_{\pi(1)} \mid \cdots \mid \mathbf{u}[n-1]\mathbf{e}_{\pi(n-1)})$. Here, $\mathbf{u} \in \mathbb{F}_q^n$, $\pi : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$ is a permutation and $\mathbf{u}[j]\mathbf{e}_{\pi(j)}$ is j -th column of \mathbf{A} . We represent the monomial matrix \mathbf{A} with the pair (π, \mathbf{u}) .

Definition 2. (Partial monomial matrix) An $n \times k$ matrix \mathbf{B} is called a partial monomial matrix if we can write the matrix $\mathbf{B} := (\mathbf{v}[0]\mathbf{e}_{\pi_*(0)} \mid \mathbf{v}[1]\mathbf{e}_{\pi_*(1)} \mid \cdots \mid \mathbf{v}[k-1]\mathbf{e}_{\pi_*(k-1)})$. Here, $n > k$, $\mathbf{v} \in \mathbb{F}_q^k$ and $\pi_* : \mathbb{Z}_k \rightarrow \mathbb{Z}_n$ is an injective mapping. We represent the partial monomial matrix \mathbf{B} with the pair (π_*, \mathbf{v}) .

We denote the set of all invertible monomial matrices of order n and the set of all partial monomial matrices of order $n \times k$ over \mathbb{F}_q by $M_n(q)$ and $M'_{n,k}(q)$ respectively.

Definition 3. (Reduced Row-Echelon form) A matrix \mathbf{A} of order $m \times n$ is said to be in Reduced Row-Echelon form (RREF) if the following conditions hold

- i. For each $0 \leq i \leq m-1, 0 \leq j \leq n-1$, if the i -th row contains the first non-zero element at j -th position, then the first non-zero element of $(i+1)$ -th row should be after the j -th position.
- ii. The first non-zero element of any non-zero row is 1.
- iii. The leading element is the only non-zero element of that column.

We can transfer any matrix \mathbf{A} to its RREF form by applying some elementary row operations [22] on the matrix \mathbf{A} , and we denote this transformation by $\text{RREF}(\mathbf{A})$. Also, note that a matrix has a unique RREF. The first non-zero elements of $\text{RREF}(\mathbf{A})$ in each row are called *pivots* and the columns that contain pivot are called *pivot column* of the matrix $\text{RREF}(\mathbf{A})$. The remaining columns are called *non-pivot columns*.

Definition 4. (Lexicographically sorted order) Let \mathbf{a} and \mathbf{b} be two vectors of the same size over the field \mathbb{F}_q . We call the vectors \mathbf{a} and \mathbf{b} are in lexicographical order if $\mathbf{a}[i] < \mathbf{b}[i]$ holds, where i is the first position where two vectors differ. We denote it as $\mathbf{a} < \mathbf{b}$. Let there be r vectors $\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{r-1}$ over the field \mathbb{F}_q . We call these vectors in lexicographically sorted order if, for any $0 \leq i, j < r$, $\mathbf{v}_i < \mathbf{v}_j$ holds whenever $i < j$.

A matrix \mathbf{G} is lexicographically sorted if its columns are in ascending lexicographical order. In this paper, the function `LexMinCol` makes each column of input matrix \mathbf{G} to lexicography sorted order by multiplying the inverse of the first non-zero element of that column and `LexSort` function is used to sort the columns of \mathbf{G} in lexicographically sorted order.

Definition 5. (Linear code) An $[n, k]$ -linear code \mathcal{C} of length n and dimension k is a linear subspace of the vector space \mathbb{F}_q^n . It can be represented by a matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$, which is called a generator matrix. For any $\mathbf{u} \in \mathbb{F}_q^k$, the generator matrix \mathbf{G} maps it to a code-word $\mathbf{u}\mathbf{G} \in \mathbb{F}_q^n$.

Definition 6. (Linear code equivalence) Let \mathcal{C} and \mathcal{C}' be two linear codes of length n and dimension k with generator matrices \mathbf{G} and \mathbf{G}' respectively. We call the codes \mathcal{C} and \mathcal{C}' linearly equivalent, if there exist matrices $\mathbf{Q} \in M_n(q)$, $\mathbf{S} \in GL_k(q)$ such that $\mathbf{G}' = \mathbf{S}\mathbf{G}\mathbf{Q}$.

Definition 7. (Information Set (IS) of a Linear Code [27]) Let \mathcal{C} be a linear code with length n , and $J \subset \mathbb{Z}_n$ be a set with cardinality k . Consider \mathbf{G} as the generator matrix of code \mathcal{C} . Define J as an information set corresponding to \mathbf{G} if inverse of $\mathbf{G}[* , J]$ exists i.e., $\mathbf{G}[* , J]$ is non-singular.

2.2 LESS Signature Scheme

The signature scheme LESS is based on the hardness of the Linear-code Equivalence Problem (LEP). LESS signature [35] uses a 3-round interactive sigma protocol [16] between a prover and a verifier to establish the message's authenticity and the Fiat Shamir transformation [1] to transform this interactive protocol into a signature scheme. In this section, we describe the key generation and the signature algorithm of the digital signature LESS as it is most relevant to our work. Meanwhile, the verification algorithm is described in Appendix B. The description of the LESS signature involves some additional functions that we describe below.

- `CSPRNG(seed, ·)`: This is a pseudo-random number generator, which takes a seed as input and outputs a pseudo-random string. The resulting output can be formatted according to preference, either as a string of seed values or a matrix. The uses of the function as `CSPRNG(seed, SRREF)`, `CSPRNG(seed, St,w)` and `CSPRNG(seed, Mn(q))` represents sampling a generator matrix in RREF, sampling the fixed weight digest vector and sampling a monomial matrix, respectively using the provided *seed*.

- **SeedTree**(*seed*, *salt*): This function generates a tree of height $\lceil \log t \rceil$. It begins with λ bit input *seed* and uses the **CSPRNG** function to generate 2λ bits. This long string is divided into two parts: the first λ bits are used for the left child and the last λ bits for the right child. The bits corresponding to each child are again fed into the **CSPRNG** with *salt* to generate the next layer of the nodes in the tree. This process is repeated until the tree with height $\lceil \log t \rceil$ is constructed.
- **PrepareDigestInput**(\mathbf{G} , \mathbf{Q}'): This function takes the matrices \mathbf{G} which is in RREF and a monomial matrix \mathbf{Q}' as inputs. Then computes \mathbf{G}' as $(\mathbf{G}', \textit{pivot_column}) = \text{RREF}(\mathbf{G}\mathbf{Q}'^T)$. Let $J = \{\alpha_0, \alpha_1, \dots, \alpha_{k-1}\}$ be the set of pivot column indices, which is essentially the information set (IS) of \mathbf{G}' . Then, compute the partial monomial matrix $\overline{\mathbf{Q}}'$ and the matrix $\overline{\mathbf{V}}'$ as $\overline{\mathbf{Q}}' = \mathbf{Q}'^T[* , J]$ and $\overline{\mathbf{V}}' = \text{LexSort}(\text{LexMinCol}(\mathbf{G}'[* , J^c]))$. After this computation, this function returns the partial monomial matrix $\overline{\mathbf{Q}}'$ and the matrix $\overline{\mathbf{V}}'$ as outputs.
- **SeedTreePaths**(*seed*, *f*): Given a seed tree *seed* and a binary string *f* representing the leaves to be disclosed, this procedure derives which nodes of the seed tree should be disclosed so that the verifier can rebuild all the leaves which have been marked by the binary string. A detailed description of this function is given in Algorithm 4.
- **CompressRREF** and **CompressMono**: **CompressRREF** function is used to compress a matrix \mathbf{G} in RREF, and similarly **CompressMono** is used to compress a monomial matrix. Each compression procedure have corresponding expansion procedure that converts the compressed information to its proper matrix form. Therefore, we can assume using or not using these function does not affect the functionality of key generation, signing or verification of LESS.

Algorithm 1. LESS.KeyGen(λ) [4, 7]

Input: None

Output: SK = (*MSEED*, *gseed*), PK = (*gseed*, $\mathbf{G}_1, \dots, \mathbf{G}_{s-1}$)

- 1: $MSEED \xleftarrow{\$} \{0, 1\}^\lambda$
 - 2: $mseed \leftarrow \text{CSPRNG}(MSEED) \in \{0, 1\}^{(s-1)\lambda}$
 - 3: $gseed \xleftarrow{\$} \{0, 1\}^\lambda$
 - 4: $\mathbf{G}_0 \leftarrow \text{CSPRNG}(gseed, \mathbb{S}_{\text{RREF}})$
 - 5: **for** $i = 1; i < s; i = i + 1$ **do**
 - 6: $\mathbf{Q}_i \leftarrow \text{CSPRNG}(mseed[i], M_n(q))$
 - 7: $(\mathbf{G}_i, \textit{pivot_column}) \leftarrow \text{RREF}(\mathbf{G}_0(\mathbf{Q}_i^{-1})^T)$
 - 8: PK[i] $\leftarrow \text{CompressRREF}(\mathbf{G}_i, \textit{pivot_column})$
 - 9: **Return** (SK, PK)
-

Key Generation of LESS: It is presented in Algorithm 1. Given a security parameter λ , the two outputs of this algorithm are the secret key SK and

the public key PK. The first component of the secret key is the master key $MSEED \in \{0, 1\}^\lambda$. Using the CSPRNG function, the vector $mseed \in \{0, 1\}^{(s-1)\lambda}$ is generated from the $MSEED$, which contains $s - 1$ many λ -bit binary strings. Now, the i -th secret monomial matrix Q_i is generated from $mseed[i] \in \{0, 1\}^\lambda$. Note that these generated Q_i 's are all secret monomial matrices. Also, the seed $gseed$ is employed in the generation of the public matrix G_0 . The remaining part of the public key consists of the matrices G_i for $1 \leq i \leq s - 1$, which are generated using the process described in Algorithm 1¹.

Algorithm 2. LESS.Sign(m , SK)

Input: Message $m \in \mathbb{Z}_2^{len}$ and secret key $SK = (MSEED, gseed)$.

Output: The signature $\tau = (salt, cmt, TreeNode, rsp)$.

```

1:  $mseed \leftarrow \text{CSPRNG}(MSEED) \in \{0, 1\}^{(s-1)\lambda}$ 
2:  $EMSEED \xleftarrow{\$} \{0, 1\}^\lambda, salt \xleftarrow{\$} \{0, 1\}^\lambda$ 
3:  $seed \leftarrow \text{SeedTree}(EMSEED, salt)$ 
4:  $ESEED = \text{Leaf nodes of the } seed$ 
5:  $G_0 \leftarrow \text{CSPRNG}(gseed, \mathbb{S}_{\text{RREF}})$ 
6: for  $i = 0; i < t; i = i + 1$  do
7:    $\tilde{Q}_i \leftarrow \text{CSPRNG}(ESEED[i], M_n(q))$ 
8:    $(\bar{Q}_i, \bar{V}_i) \leftarrow \text{PrepareDigestInput}(G_0, \tilde{Q}_i)$ 
9:    $cmt \leftarrow H(\bar{V}_0, \dots, \bar{V}_{t-1}, m, len, salt)$ 
10:  $d \leftarrow \text{CSPRNG}(cmt, \mathbb{S}_{t,w})$ 
11: for  $i = 0; i < t; i = i + 1$  do
12:   if  $d[i] = 0$  then
13:      $f[i] = 0$ 
14:   else
15:      $f[i] = 1$ 
16:  $TreeNode \leftarrow \text{SeedTreePaths}(seed, f)$  ▷ (Alg. 4)
17:  $k = 0$ 
18: for  $i = 0; i < t; i = i + 1$  do
19:   if  $d[i] \neq 0$  then
20:      $j = d[i]$ 
21:      $Q_j \leftarrow \text{CSPRNG}(mseed[j], M_n(q))$ 
22:      $Q_k^* \leftarrow Q_j^T Q_i$ 
23:      $rsp[k] \leftarrow \text{CompressMono}(Q_k^*)$ 
24:      $k = k + 1$ 
25: Return  $\tau = (salt, cmt, TreeNode, rsp)$ 

```

Signature Algorithm of LESS: The signature algorithm shown in Algorithm 2 takes a message string m of length len and the secret key $SK = (MSEED, gseed)$ as inputs and returns a corresponding signature τ . The main

¹ For simplicity and compactness, we follow the implementation of LESS instead of the specification document.

secret key component of SK is the master seed $MSEED$. All the $s - 1$ monomial matrices Q_j are generated from the $MSEED$ and used to produce signatures. That is, instead of having information of $MSEED$, if we have the information of all of $s - 1$ monomial matrices Q_j , then we can construct the same valid signature. Therefore, these monomial matrices Q_j are considered equivalent to the secret key component $MSEED$. To reduce the signature size, the authors of LESS have incorporated a method involving tree construction. We explain this process briefly here.

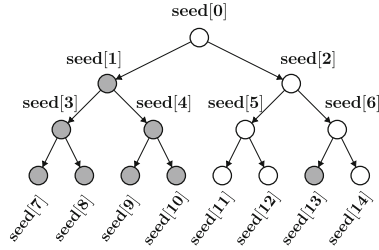


Fig. 1. Example of seed tree

First, we outline the procedure for generating a set of t ephemeral monomial matrices represented by Q_0, Q_1, \dots, Q_{t-1} through the generation of t random ephemeral seeds denoted as $ESEED[i]$ for $0 \leq i < t$. The process involves the following steps:

- Start by sampling a random master seed $EMSEED \xleftarrow{\$} \{0, 1\}^\lambda$.
- Build a tree of seed nodes using `SeedTree` procedure, with output tree named *seed*. The height and the number of leaf nodes of the output tree are $\lceil \log(t) \rceil$ and $2l = 2^{\lceil \log(t) \rceil}$ respectively where the input seed is the master seed $EMSEED$.
- Select the first t leaf nodes of the *seed* as the ephemeral seeds $ESEED[i]$, where $ESEED[i] = seed[2l - 1 + i]$ for $0 \leq i < t$.
- Using CSPRNG function Q_i is prepared for each $ESEED[i]$.

Lines 6–8 of Algorithm 2, correspond to generating the partial monomial matrices \bar{Q}_i , the matrices \bar{V}_i having the information of the non-pivot part corresponding to the matrix $G_0 \tilde{Q}_i^T$. Using the information of all \bar{V}_i matrices, message m , message length len and $salt$, the digest $\mathbf{d} \in \mathbb{Z}_s^t$ is prepared. This digest vector \mathbf{d} has fixed weight w , where weight of the vector \mathbf{d} is defined as $wt(\mathbf{d}) := |\{i : \mathbf{d}[i] \neq 0\}|$. We will briefly discuss the `SeedTreePaths` procedure in Algorithm 4, as our attack is based on exploiting this procedure. This `SeedTreePaths` (Algorithm 4) helps to reduce the size of the signature. Finally, the signature will return $Q_{\mathbf{d}[i]}^T \bar{Q}_i$ whenever $\mathbf{d}[i] \neq 0$, and it also reveals the seed nodes so that \tilde{Q}_i can be generated from the revealed seed nodes for all

i such that $\mathbf{d}[i] = 0$. Note that whenever we try to return \tilde{Q}_i , it is enough to return $\mathbf{ESEED}[i]$'s instead. Also, having the information of any ancestor node of the seed $\mathbf{ESEED}[i](= \mathit{seed}[2l - 1 + i])$, we can get the information of $\mathbf{ESEED}[i]$. This is the idea behind the minimization of the number of seeds that are to be sent. This minimized set is returned as $\mathbf{TreeNode}$. Consider the example in Fig. 1, where the leaf nodes are $\mathbf{ESEED}[i]$'s and the shaded leaf nodes represent all those positions where \mathbf{d} takes the value 0 *i.e.*, these are the $\mathbf{ESEED}[i]$'s that are to be revealed. Observe that revealing only $\mathbf{TreeNode} = (\mathit{seed}[1], \mathit{seed}[13])$ is enough, as the required $\mathbf{ESEED}[i]$'s can be regenerated at the time of verification. Consequently, this minimizes the signature size.

Now, in lines 18–24 of Algorithm 2, the \mathbf{rsp} is prepared by appending the partial monomial matrices $Q_{\mathbf{d}[i]}^T \overline{Q}_i$ for all i such that $\mathbf{d}[i]$ is non-zero. Since the length and the weight of the fixed weight digest \mathbf{d} are t and w respectively, the vector \mathbf{d} has exactly w many non-zero elements and $t - w$ many zero elements. Therefore, the signature will contain the component \mathbf{rsp} having exactly w many matrices of the form $Q_{\mathbf{d}[i]}^T \overline{Q}_i$. After all of these computations, $(\mathit{salt}, \mathit{cmt}, \mathbf{TreeNode}, \mathbf{rsp})$ is generated as the signature.

2.3 Parameter Set

There are three security levels of LESS [35] and their corresponding parameter sets, which are shown in Table 1. Here, the code parameters are given by n : the length of the code, k : the dimension of the code, q : prime modulus corresponding to the finite field \mathbb{F}_q , $2l$: the number of leaf nodes of the seed tree, where $2l = 2^{\lceil \log t \rceil}$, t : the length of the digest \mathbf{d} , w : the fixed weight of the digest \mathbf{d} and s : $s - 1$ is the number of secret monomial matrices. According to the LESS documentation [35], multiple parameter sets are defined for each security level of LESS, and the optimization criteria for each of these parameter sets are different. The “b” version (e.g., LESS-1b) refers to the parameter set with balanced public key and signature size, the “s” (e.g., LESS-1s) version refers to the parameter set with smaller signature size, and the “i” (only LESS-1i) version refers to the parameter set with intermediate public key and signature size.

3 Our Work: Fault Analysis of LESS

One of the strongest physical attacks on the digital signature schemes is to recover the secret or signing key, as the adversary can compute any valid message and signature pair using the recovered signing key. In general, only the key generation and the signing algorithm involve the secret key. However, only the signing algorithm uses the long-term secret key (the same secret key is used multiple times), making it most suitable for performing a physical attack [9, 18, 29, 38]. In this work, our objective is to mount a fault attack on the zero-knowledge based digital signature schemes. In this attack model, the adversary would query the faulted signature oracle (which outputs a signature with some

Table 1. Parameter set of LESS [35] for different security levels

Security level	Parameter set	Parameters						Public key (PK) (KiB)	Signature (τ) (KiB)		
		n	k	q	l	t	w			s	
1	LESS-1b					247	30	2	13.7	8.1	
	LESS-1i	252	126	127	128	244	20	4	41.1	6.1	
	LESS-1s					198	17	8	95.9	5.2	
3	LESS-3b					759	33	2	34.5	18.4	
	LESS-3s	400	200	127	512	895	26	3	68.9	14.1	
5	LESS-5b					1024	1352	40	2	64.6	32.5
	LESS-5s	548	274	127	512	907	37	3	129.0	26.1	

injected faults) multiple times. In this section, we will progressively describe our fault attack strategy to recover the secret monomial matrices for the LESS signature scheme. Later in Sect. 4, we show that the same attack strategy can be employed in other zero-knowledge based signature schemes, such as CROSS, to recover the signing key.

3.1 An Observation on LESS

LESS signature algorithm presented in Algorithm 2 returns either the information of the monomial matrix \tilde{Q}_j or the multiplication $Q_{d[j]}^T \overline{Q}_j$ for any $j \in \mathbb{Z}_t$. Here, \overline{Q}_j is a *partial monomial* matrix that is generated from the matrix \tilde{Q}_j by using the `PrepareDigestInput` function. If we manage to get a pair $(\tilde{Q}_j, Q_{d[j]}^T \overline{Q}_j)$ for some $d[j] \neq 0$, then we can construct the pair $(\overline{Q}_j, Q_{d[j]}^T \overline{Q}_j)$. This pair $(\overline{Q}_j, Q_{d[j]}^T \overline{Q}_j)$ leaks some information of matrix $Q_{d[j]}^T$ that is directly follows from the following lemma.

Lemma 1. *Let $\mathbf{A} = (\pi, \mathbf{u}) \in M_n(q)$ be a monomial matrix and $\mathbf{B} = (\pi', \mathbf{u}') \in M'_{n,k}(q)$ be a partial monomial matrix. Let $\mathbf{C} = (\pi'', \mathbf{u}'') \in M'_{n,k}(q)$ be the partial monomial matrix defined by $\mathbf{C} = \mathbf{A}^T \mathbf{B}$. Given the matrices \mathbf{B} and \mathbf{C} , we can compute exactly k many columns of the monomial matrix \mathbf{A}^T . More specifically, for all $0 \leq j < k$, we can compute $\pi^{-1}(\pi'(j))$ and $\mathbf{u}[\pi^{-1}(\pi'(j))]$.*

Proof. For the monomial matrix \mathbf{A} represented by (π, \mathbf{u}) , the transpose of \mathbf{A} is the following matrix

$$\mathbf{A}^T = [\mathbf{u}[\pi^{-1}(0)]\mathbf{e}_{\pi^{-1}(0)} \mid \mathbf{u}[\pi^{-1}(1)]\mathbf{e}_{\pi^{-1}(1)} \mid \cdots \mid \mathbf{u}[\pi^{-1}(n-1)]\mathbf{e}_{\pi^{-1}(n-1)}]$$

The multiplication of the monomial matrix \mathbf{A}^T with the partial monomial matrix \mathbf{B} is given by

$$\mathbf{A}^T \mathbf{B} = [\mathbf{u}[\pi^{-1}(\pi'(0))]\mathbf{u}'[0]\mathbf{e}_{\pi^{-1}(\pi'(0))} \mid \cdots \mid \mathbf{u}[\pi^{-1}(\pi'(k-1))]\mathbf{u}'[k-1]\mathbf{e}_{\pi^{-1}(\pi'(k-1))}]$$

Since $\mathbf{C} = \mathbf{A}^T \mathbf{B}$, so for all $0 \leq j < k$ we have $\mathbf{C}[* , j] = (\mathbf{A}^T \mathbf{B})[* , j]$, which implies $\mathbf{u}''[j] \mathbf{e}_{\pi''(j)} = \mathbf{u}[\pi^{-1}(\pi'(j))] \mathbf{u}'[j] \mathbf{e}_{\pi^{-1}(\pi'_*(j))}$. This gives us the following

$$\begin{aligned} \mathbf{u}''[j] &= \mathbf{u}[\pi^{-1}(\pi'(j))] \mathbf{u}'[j] \\ \pi''(j) &= \pi^{-1}(\pi'(j)) \end{aligned}$$

Since \mathbf{B} and \mathbf{C} are known, we have the information of each $\pi'(j)$, $\pi''(j)$, $\mathbf{u}'[j]$ and $\mathbf{u}''[j]$ where $0 \leq j < k$. Therefore for all $0 \leq j < k$ we have,

$$\begin{aligned} \mathbf{u}[\pi^{-1}(\pi'(j))] &= \mathbf{u}''[j] (\mathbf{u}'[j])^{-1} \\ \pi^{-1}(\pi'(j)) &= \pi''(j) \end{aligned} \quad (1)$$

Note that we have computed $\pi'(j)$ -th column of the matrix \mathbf{A}^T for all $0 \leq j < k$. \square

For simplicity, in this part, we will use consider the matrices $\tilde{\mathbf{Q}}_j$, $\bar{\mathbf{Q}}_j$ and $\mathbf{Q}_{d[j]}$ as the matrices $\tilde{\mathbf{Q}}$, $\bar{\mathbf{Q}}$ and \mathbf{Q} respectively. Recall the `prepareDigestInput` function, it was taking \mathbf{G}_0 and a monomial matrix $\tilde{\mathbf{Q}} = (\tilde{\pi}, \tilde{\mathbf{v}})$ as input and $\bar{\mathbf{Q}}$ is one of the outputs of the function. The $\bar{\mathbf{Q}}$ is computed in a way that $\mathbf{G}_0 \bar{\mathbf{Q}} = \mathbf{G}_0 (\tilde{\mathbf{Q}})^T [* , J^\dagger]$, where J^\dagger is an IS of $\mathbf{G}_0 (\tilde{\mathbf{Q}})^T$. From the definition of IS, we can say that $\mathbf{G}_0 \bar{\mathbf{Q}}$ is a non-singular matrix. Observe that,

$$\mathbf{G}_0 \bar{\mathbf{Q}} = [\bar{\mathbf{v}}_0 \cdot \mathbf{g}_{\pi(0)} \mid \bar{\mathbf{v}}_1 \cdot \mathbf{g}_{\pi(1)} \mid \cdots \mid \bar{\mathbf{v}}_{k-1} \cdot \mathbf{g}_{\pi(k-1)}] \quad (2)$$

Where $\bar{\mathbf{Q}}$ is a partial monomial matrix represented by $(\bar{\pi}, \bar{\mathbf{v}})$. Since the matrix representation in Eq. 2 is non-singular, the set $J = \{\bar{\pi}(i) : i \in \mathbb{Z}_k\}$ is the IS of \mathbf{G}_0 .

Now consider we are given the pair $(\tilde{\mathbf{Q}}, \mathbf{Q}^T \bar{\mathbf{Q}})$, where \mathbf{Q} represented by (π, \mathbf{v}) and $\bar{\mathbf{Q}}$ is generated from $\tilde{\mathbf{Q}}$ using the function `prepareDigestInput`. Now, $\mathbf{Q}^T \bar{\mathbf{Q}}$ is a partial monomial matrix and let it be represented by (π_*, \mathbf{v}_*) then from Lemma 1, we can write that for any $j \in \mathbb{Z}_k$

$$\begin{aligned} \pi^{-1}(\bar{\pi}(i)) &= \pi_*(i) \\ \mathbf{v}[\pi^{-1}(\bar{\pi}(i))] &= \mathbf{v}_*[i] (\bar{\mathbf{v}}[i])^{-1}. \end{aligned} \quad (3)$$

This Eq. 3 gives us the partially recovered secret *i.e.* only k many columns of \mathbf{Q}^T . According to the definition of $\bar{\pi}$, the set $\{\bar{\pi}(i) : i \in \mathbb{Z}_k\}$ is the set J which is the information set of \mathbf{G}_0 . Now, if \mathbf{Q} is a secret monomial then from the key generation of LESS, we can say that $\hat{\mathbf{G}} = \text{RREF}(\mathbf{G}_0 (\mathbf{Q}^T)^{-1})$ is a part of the public key. We can further write $\hat{\mathbf{G}} = \mathbf{S} \mathbf{G}_0 (\mathbf{Q}^T)^{-1}$ for some non-singular matrix \mathbf{S} . Consider $\hat{\mathbf{G}} = [\hat{\mathbf{g}}_0 \mid \hat{\mathbf{g}}_1 \mid \cdots \mid \hat{\mathbf{g}}_{n-1}]$ then for all $i \in \mathbb{Z}_n$ we have $\hat{\mathbf{g}}_i = \mathbf{S} \cdot ((\mathbf{v}[i])^{-1} \cdot \mathbf{g}_{\pi(i)})$ which implies that for all $i \in \mathbb{Z}_n$,

$$\hat{\mathbf{g}}_{\pi^{-1}(i)} = \mathbf{S} \cdot ((\mathbf{v}[\pi^{-1}(i)])^{-1} \cdot \mathbf{g}_i) \quad (4)$$

Consider that the set J have the elements j_0, j_1, \dots, j_{k-1} , and we take the matrix $\mathbf{G}^* = [\hat{\mathbf{g}}_{\pi^{-1}(j_0)} \mid \hat{\mathbf{g}}_{\pi^{-1}(j_1)} \mid \cdots \mid \hat{\mathbf{g}}_{\pi^{-1}(j_{k-1})}]$ and also take the matrix

$$\mathbf{G}' = [(\mathbf{v}[\pi^{-1}(j_0)])^{-1} \cdot \mathbf{g}_{j_0} \mid (\mathbf{v}[\pi^{-1}(j_1)])^{-1} \cdot \mathbf{g}_{j_1} \mid \cdots \mid (\mathbf{v}[\pi^{-1}(j_{k-1})])^{-1} \cdot \mathbf{g}_{j_{k-1}}]$$

From Eq. 4, we have $\mathbf{G}^* = \mathbf{S}\mathbf{G}'$ and since J is an IS of \mathbf{G}_0 , so \mathbf{G}' is a non-singular matrix. Also \mathbf{G}' and \mathbf{G}^* are both computable as for each $j \in J$, $\pi^{-1}(j)$ and $\mathbf{v}[\pi^{-1}(j)]$ are already recovered. Therefore, we can compute $\mathbf{S} = \mathbf{G}^* \cdot (\mathbf{G}')^{-1}$. Finally, we have $\mathbf{S}^{-1}\widehat{\mathbf{G}} = \mathbf{G}_0(\mathbf{Q}^T)^{-1}$, where \mathbf{S} , \mathbf{G}_0 and $\widehat{\mathbf{G}}$ are known. Using Algorithm 3, we can recover the full secret.

Algorithm 3. `getColumnPermutation`($\widehat{\mathbf{G}}, \mathbf{G}_0, \mathbf{S}$)

Input: The partially recovered secret $\pi : J_* \rightarrow J$ and $\mathbf{v}[j] \forall j \in J_*$, where $J_* = \{\pi^{-1}(i) : i \in J\}$, public information \mathbf{G}_0 and $\widehat{\mathbf{G}}$, recovered matrix \mathbf{S}

Output: Outputs rest of the secret $\pi : J_* \rightarrow J^c$ and $\mathbf{v}[j] \forall j \in J^c$

```

1:  $[g_0 \mid g_1 \mid \dots \mid g_{n-1}] \leftarrow \mathbf{G}_0$ 
2:  $[\widehat{g}_0 \mid \widehat{g}_1 \mid \dots \mid \widehat{g}_{n-1}] \leftarrow \widehat{\mathbf{G}}$ 
3: for  $j \in J^c$  do
4:   for  $i \in J_*^c$  do
5:     for  $a \in \mathbb{F}_q$  do
6:       if  $g_j = a \cdot (\mathbf{S}^{-1}\widehat{g}_i)$  then
7:         assign  $\pi(i) \leftarrow j$ 
8:         assign  $\mathbf{v}[i] \leftarrow a$ 

```

We can conclude that from one pair $(\widetilde{\mathbf{Q}}_j, \mathbf{Q}_{d[j]}^T \overline{\mathbf{Q}}_j)$, we can recover the secret monomial matrix $\mathbf{Q}_{d[j]}^T$, where $d[j] \neq 0$. However, we will not receive the pair $(\widetilde{\mathbf{Q}}_j, \mathbf{Q}_{d[j]}^T \overline{\mathbf{Q}}_j)$ if the signatures are generated by executing the signing algorithm properly. Therefore, we must find strategies to disrupt the normal flow of execution to help us get such pairs. Also, note that, if the number of secret monomial matrices ($s-1$) is greater than one, then receiving only one such pair is not enough to retrieve all the secret monomials. So, we may require multiple faulted signatures to receive several such pairs and finally recover all the secret monomial matrices. All of these analysis are briefly described in the later sections.

3.2 Identification of Attack Surfaces

As we observed that having one pair of the form $(\widetilde{\mathbf{Q}}_j, \mathbf{Q}_{d[j]}^T \overline{\mathbf{Q}}_j)$ is enough to recover the secret matrix $\mathbf{Q}_{d[j]}^T$, where $d[j] \neq 0$. Also, observe that, in LESS, there are $s-1$ secret monomial matrices \mathbf{Q}_i for $1 \leq i \leq s-1$, and t ephemeral monomial matrices $\widetilde{\mathbf{Q}}_j$ for $0 \leq j < t$ as described in Sect. 2.2. Hence, our goal is to find at least one pair of the form $(\widetilde{\mathbf{Q}}_j, \mathbf{Q}_{d[j]}^T \overline{\mathbf{Q}}_j)$, where $1 \leq d[j] \leq s-1$ and $0 \leq j < t$ by manipulating the signing algorithm.

Note that, LESS is a code-based signature scheme based on the sigma-protocol with Fiat-Shamir transformation. In Algorithm 2, the signer generates the random challenge \mathbf{d} (fixed weight digest), from commitment (cmt) using the pseudo-random function CSPRNG. Any fault injection before the challenge generation may modify the challenge value, but that is an output of a pseudo-random

function. This would not help, as we need to recover the secret key. Therefore, we have targeted to inject a fault after the generation of \mathbf{d} .

Modification of the Vector \mathbf{d} : As we can see from Algorithm 2, the digest \mathbf{d} ($\mathbf{d}[i]$ for $0 \leq i < t$) value decides whether \tilde{Q}_i is revealed or $Q_{\mathbf{d}[i]}^T \tilde{Q}_i$ is revealed. Therefore, the most obvious target for fault injection is the digest \mathbf{d} to reveal both \tilde{Q}_i and $Q_{\mathbf{d}[i]}^T \tilde{Q}_i$ for some i . If we modify some value $\mathbf{d}[i]$ of \mathbf{d} (line 11 in Algorithm 2) from 0 to some non-zero value r by injecting fault, then we will get the information of $Q_r^T \tilde{Q}_i$ instead of getting information of \tilde{Q}_i . Similarly, if we change the value of $\mathbf{d}[i]$ from non-zero value r to 0, then we will get the information of \tilde{Q}_i instead of getting information about $Q_r^T \tilde{Q}_i$. In both cases, we do not receive \tilde{Q}_i and $Q_r^T \tilde{Q}_i$ together. Therefore, modifying the \mathbf{d} value does not satisfy our purpose.

Algorithm 4. SeedTreePaths

Input: The *Seed Tree seed* and the vector \mathbf{f} .

Output: Outputs *TreeNode* a subset of *Seed Tree* which consists only the *seed*'s that does not correspond to $\mathbf{f}[i] = 1$.

```

1: for  $i = 0; i < 4l - 1; i = i + 1$  do
2:    $\mathbf{x}[i] = 0$ 
3:  $\mathbf{x} \leftarrow \text{compute\_seeds\_to\_publish}(\mathbf{f}, \mathbf{x})$  ▷ (Alg. 5)
4:  $j = 0$ 
5: for  $i = 0; i < 4l - 1; i = i + 1$  do
6:   if ( $\mathbf{x}[i] = 0$  and  $\mathbf{x}[\text{Parent}(i)] = 1$ ) then
7:     TreeNode $[j] = \text{seed}[i]$ 
8:      $j = j + 1$ 
9: return TreeNode

```

One might think of using the cases $\mathbf{d}[i] = 0$ bypassing the check $\mathbf{d}[i] \neq 0$ (line 19) using a fault. However, $mseed[0]$ does not exist and might cause an error during execution. Therefore, modifying anything from lines 18–24 would not benefit us. Now, we analyse the remaining steps (lines 11–16) of Algorithm 2. In these steps, we can modify the value of the vector \mathbf{f} . Also, the *SeedTreePaths* algorithm is another potential candidate for fault injection, which is presented in Algorithm 4. It uses an auxiliary function *compute_seeds_to_publish* described in Algorithm 5. In the *SeedTreePaths* procedure, a tree \mathbf{x} of size $4l - 1$ is initialized with all zero. We call this tree as *Reference Tree*. In Algorithm 5, the values of the leaf nodes of the *Reference Tree* are updated according to the \mathbf{f} i.e., $\mathbf{x}[2l-1+i]$ are assigned the value $\mathbf{f}[i]$ for all $0 \leq i < t$. The remaining nodes of the *Reference Tree* are assigned the value using the formula $\mathbf{x}[i] = \mathbf{x}[2i+1] \vee \mathbf{x}[2i+2]$, signifying that if either child has a value of 1, the corresponding parent will be assigned 1. In this way, the value of the *Reference Tree* \mathbf{x} has been updated in a bottom-up approach. Now, some locations in *Seed Tree* are to be published as *TreeNode* with the help of the *Reference Tree*. Algorithm 4 checks if the i -th

Algorithm 5. compute_seeds_to_publish

Input: A vector \mathbf{f} of size t and the Reference Tree \mathbf{x} .

Output: Modified Reference Tree \mathbf{x} .

- 1: **for** $i = 0; i < t; i = i + 1$ **do**
 - 2: $\mathbf{x}[2l - 1 + i] = \mathbf{f}[i]$
 - 3: **for** $i = 2l - 2; i \geq 0; i = i - 1$ **do**
 - 4: $\mathbf{x}[i] = \mathbf{x}[2i + 1] \vee \mathbf{x}[2i + 2]$
 - 5: **return** \mathbf{x}
-

node of Reference Tree $\mathbf{x}[i]$ is zero and its parent $\mathbf{x}[Parent(i)]$ is 1, where the function $Parent(\cdot)$ is defined as follows:

$$Parent(i) = \begin{cases} 0 & \text{if } i = 0 \\ \lfloor \frac{i-1}{2} \rfloor & \text{otherwise} \end{cases}$$

If the validity check is satisfied, then $seed[i]$, the i -th node of the Seed Tree is appended to **TreeNode**.

Example 1. In Fig. 2, we have given an example for leaf nodes $2l = 8$ and the vector \mathbf{d} is chosen as $(0, 3, 1, 1, 0, 0, 0, 0)$. Then, the vector \mathbf{f} will be $(0, 1, 1, 1, 0, 0, 0, 0)$. From Fig. 2, we can see that for $i = 2, 7$ the condition “ $\mathbf{x}[i] = 0$ and $\mathbf{x}[Parent(i)] = 1$ ” is satisfied. Therefore, the vector $\mathbf{TreeNode} = (seed[2], seed[7])$ and $\mathbf{rsp} = (Q_{d[1]}^T \overline{Q}_1, Q_{d[2]}^T \overline{Q}_2, Q_{d[3]}^T \overline{Q}_3) = (Q_3^T \overline{Q}_1, Q_1^T \overline{Q}_2, Q_1^T \overline{Q}_3)$ will be extracted from Seed Tree is revealed at the end. From the seeds $seed[2]$ $seed[7]$, we can compute the leaf seeds $seed[7], seed[11], seed[12], seed[13], seed[14]$ which are equals to the leaf seeds $ESEED[0], ESEED[4], ESEED[5], ESEED[6], ESEED[7]$ respec-

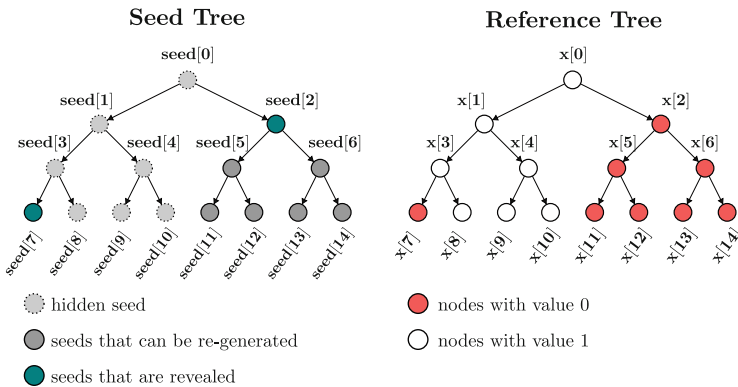


Fig. 2. Example for extraction of **TreeNode** from Seed Tree using Reference Tree, following Algorithm 4

tively. From these seeds, we can compute the matrices $\widetilde{Q}_0, \widetilde{Q}_4, \widetilde{Q}_5, \widetilde{Q}_6, \widetilde{Q}_7$. From the output of LESS_Sign algorithm, we will get either \widetilde{Q}_j or $Q_{d[j]}^T \widetilde{Q}_j$.

As we can observe from Algorithm 4, the seeds from *Seed Tree* that are revealed as *TreeNode* are directly associated with the values in \mathbf{f} and the *Reference Tree* \mathbf{x} . Therefore, we can try injecting faults in various locations of \mathbf{f} or \mathbf{x} .

Modification of any node of the Reference Tree \mathbf{x} : Here, we investigate the effect of modification of some fixed i -th value of the tree \mathbf{x} in Algorithm 5. Without loss of generality, assume $\mathbf{x}[i_0], \mathbf{x}[i_1], \dots, \mathbf{x}[i_{r-1}]$ be the leaf nodes of the subtree with root node $\mathbf{x}[i]$, where $r \geq 1$. Now, suppose we inject a fault in the signature algorithm to modify the value of i -th node of the *Reference Tree* \mathbf{x} . In that case, the signature algorithm will give us the faulted signature. However, even if we try to inject a fault in a physical machine, the fault can only occur with a certain probability. If we assume that the fault injection is successful, even then, there are several cases:

- **Case 1:** The node $\mathbf{x}[i]$ is 0 in the non-faulted case. In this case, since the actual value $\mathbf{x}[i]$ is 0, all the leaf nodes in the subtree rooted at $\mathbf{x}[i]$ must be zero. Hence, the vectors \mathbf{f} and \mathbf{d} do not have any non-zero value at the positions corresponding to the leaf nodes $\mathbf{x}[i_0], \mathbf{x}[i_1], \dots, \mathbf{x}[i_{r-1}]$. Therefore, the \mathbf{rsp} does not contain multiplication of any secret monomial matrix with the partial monomial matrix \widetilde{Q}_{i_j-2l+1} , where $0 \leq j < r$ i.e., we can not get any information about the secret matrices.
- **Case 2:** The node $\mathbf{x}[i]$ is 1 in the non-faulted case, and after the fault injection, it has changed to 0. Since the *Reference Tree* is updated in a bottom-up approach, the modification of the i -th node $\mathbf{x}[i]$ may affect the ancestors of $\mathbf{x}[i]$. Consequently, it may change the root node $\mathbf{x}[0]$. In this case, assume that it changes the value of the root node $\mathbf{x}[0]$ to 0. This case can occur only if all non-zero leaves fall under the subtree rooted at $\mathbf{x}[i]$. Since the value of the root node is zero, all the ancestors of $\mathbf{x}[i]$ including $\mathbf{x}[0]$ are zero. Therefore, neither $\mathbf{seed}[i]$ nor any of its ancestors in *Seed Tree* is released because the Algorithm 4 requires the parent of $\mathbf{x}[j]$ to be 1 if we want to release the $\mathbf{seed}[j]$, i.e. such fault does not provide any advantage to us. Therefore, the nodes in the subtree rooted at $\mathbf{x}[i]$ do not affect the fault injection, so no extra information can be achieved from the released seeds corresponding to this subtree.

Example 2. We consider the fixed digest vector \mathbf{d} , \mathbf{f} , and the *Reference Tree* \mathbf{x} of Example 1 in a non-faulted scenario. We modify the value of $\mathbf{x}[1]$ from $1 \rightarrow 0$ that changes the value of $\mathbf{x}[0]$ from $1 \rightarrow 0$. Figure 3 represents the *Reference Tree* and the related node of *Seed Tree* in faulted case. In this case, only $\mathbf{x}[7]$ satisfies the condition “ $\mathbf{x}[7] = 0$ and $\mathbf{x}[\text{Parent}(7)] = 1$ ”. Therefore, *TreeNode* will be ($\mathbf{seed}[7]$) and $\mathbf{rsp} = (Q_{d[1]}^T \widetilde{Q}_1, Q_{d[2]}^T \widetilde{Q}_2, Q_{d[3]}^T \widetilde{Q}_3) = (Q_3^T \widetilde{Q}_1, Q_1^T \widetilde{Q}_2, Q_1^T \widetilde{Q}_3)$. None of the monomial matrices $\widetilde{Q}_1, \widetilde{Q}_2, \widetilde{Q}_3$ can be generated from $\mathbf{seed}[7]$. Therefore, we are unable to recover any secret key-related information from this faulted signature.

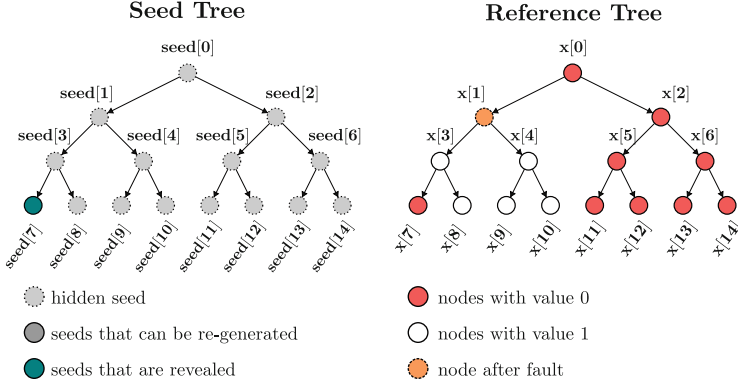


Fig. 3. Example of Case 2

- **Case 3:** The node $x[i]$ is 1 in the non-faulted case. After the fault injection, it has changed to 0, but $x[0]$ remains 1. Since the actual value of $x[i]$ is 1, there exists some leaf node $x[i_j]$ such that $x[i_j] = 1$. Therefore, it follows that $f[i_j - 2l + 1]$ is non-zero and consequently $d[i_j - 2l + 1]$ is also non-zero. Without loss of generality, assume that $i_j - 2l + 1 = k'$ then rsp contains $Q_{d[k']}^T \bar{Q}_{k'}$. Also, since the faulted value of $x[i]$ is 0 and $x[0] = 1$, so $TreeNode$ will contain either $seed[i]$ or any of its ancestors in $Seed Tree$ from which we can generate the leaf node $seed[i_j]$ of the subtree rooted at $seed[i]$. Hence, the ephemeral key $ESEED[k']$ and consequently the monomial matrix $\tilde{Q}_{k'}$ can be generated. Therefore, we retrieve the pair $(\bar{Q}_{k'}, Q_{d[k']}^T \bar{Q}_{k'})$.

Example 3. We consider the fixed digest vector d, f , and the *Reference Tree* x of Example 1 in a non-faulted scenario. We modify the value of $x[3]$ from $1 \rightarrow 0$ that does not change the value of $x[0]$. Figure 4 represents the *Reference Tree* and the related node of *Seed Tree* in the faulted case. From this Fig. 4 we can see that for $i = 2, 3$ the condition “ $x[i] = 0$ and $x[Parent(i)] = 1$ ” is satisfied. Therefore, $TreeNode$ will be $(seed[2], seed[3])$ and $rsp = (Q_3^T \bar{Q}_1, Q_1^T \bar{Q}_2, Q_1^T \bar{Q}_3)$. Now $seed[3]$ is contained in the signature component *Reference Tree* that we can generate the seed $seed[8]$ to the $8 - 2l + 1 = 8 - 8 + 1 = 1$ -st monomial matrix \tilde{Q}_1 . So, from this faulted signature, we found the pair $(\tilde{Q}_1, Q_3^T \bar{Q}_1)$ that help us find the information of the matrix Q_3^T .

Modification of the Vector f : The vector f is computed by using the fixed digest vector d . If the i -th element of d holds a non-zero value, $f[i]$ is assigned the value of 1; otherwise, it is set to zero. If we modify the i -th value $f[i]$ by injecting fault, then the $(2l - 1 + i)$ -th leaf node $x[2l - 1 + i]$ of the *Reference Tree* will be changed. The effect of this fault will be the same as the above modification of any leaf node of the *Reference Tree* x .

From the above, we can observe that the attack surfaces are different as in the second attack component, we change the value of any $f[i]$, and in the first

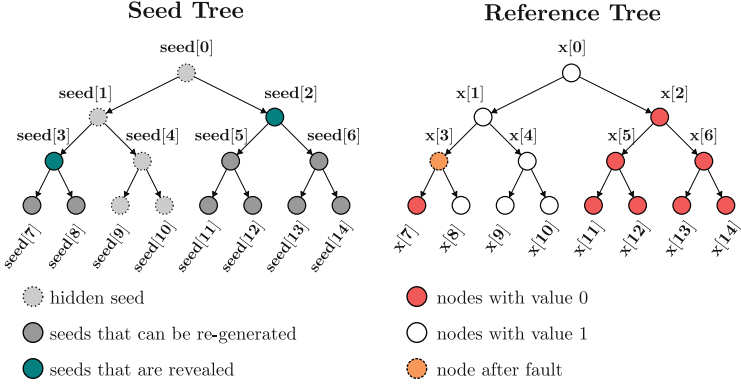


Fig. 4. Example of Case 3

attack component, we change the value of any $\mathbf{x}[i]$. However, we can say that modifying the value of any value $\mathbf{f}[i]$ is imposing the same effect as modifying the corresponding leaf node of $\mathbf{x}[2l - 1 + i]$. Therefore, from now onwards, we only discuss the modification of any node of the *Reference Tree* \mathbf{x} .

3.3 Fault Models

In this section, we describe the fault models that will help us to recover the secret key. Our attack just requires changing a bit ($1 \rightarrow 0$ for LESS). Here we discussed in detailed how each fault model can be utilize to realize our attack. Mainly, our fault assumptions can be realized by “skipping one condition check” or “forcing one data corruption in \mathbf{f} or \mathbf{x} ”. We assume that the faulted location is arbitrary but known to the attacker.

Skip the Validity Check Condition in Algorithm 4: If we skip the check “ $\mathbf{x}[i] = 0$ and $\mathbf{x}[\text{Parent}(i)] = 1$ ” in Algorithm 4 for a fixed i , then $\text{seed}[i]$ will always contained in *TreeNode*. Without loss of generality, let $\text{seed}[i_0], \dots, \text{seed}[i_{r-1}]$ be the leaf nodes of the subtree rooted the node $\text{seed}[i]$ of *Seed Tree* and $\mathbf{x}[i_0], \dots, \mathbf{x}[i_{r-1}]$ be the corresponding leaf nodes of the subtree rooted the node $\mathbf{x}[i]$ of the *Reference Tree*. If $\mathbf{x}[i] = 1$, then there exists the leaf node say $\mathbf{x}[i_{j'}]$, where $0 \leq j' < r$ and $\mathbf{x}[i_{j'}] = 1$. This implies $\mathbf{f}[i_{j'} - 2l + 1] = 1$ and so, $\mathbf{d}[i_{j'} - 2l + 1]$ must be non-zero. Therefore, \mathbf{rsp} must contain the matrix multiplication $\mathbf{Q}_d^T[i_{j'} - 2l + 1] \bar{\mathbf{Q}}_{i_{j'} - 2l + 1}$. Since $\mathbf{d}[i_{j'} - 2l + 1]$ is non-zero, we are not supposed to have the information about the ephemeral matrix $\bar{\mathbf{Q}}_{i_{j'} - 2l + 1}$ in non-faulted case. However, from the $\text{seed}[i]$, we can generate all leaf nodes of the subtree rooted in this node. Now, $\text{seed}[i_{j'}]$ is the $i_{j'} - 2l + 1$ -th leaf node **ESEED** $[i_{j'} - 2l + 1]$ of the *Seed Tree*, and that helps us find the ephemeral monomial matrix $\bar{\mathbf{Q}}_{i_{j'} - 2l + 1}$. So, in this case, we can find some information of secret monomial matrix $\mathbf{Q}_d^T[i_{j'} - 2l + 1]$ from the pair

$(\tilde{Q}_{i_{j'}-2l+1}, Q_{d[i_{j'}-2l+1]}^T \bar{Q}_{i_{j'}-2l+1})$. This is a model that we can use to mount the attack.

Previously, many works [9, 28] have shown that instruction skips can be easily done with clock glitches, and the fault happens with very high probability. Mainly, in these works they have skipped the condition check instructions and store instructions. Recently, Keita et al. in [39] bypassed the validity check in the decapsulation procedure in post-quantum the key-encapsulation mechanism Kyber [3]. However, one may argue that skipping the validity check is the most important part as if we can skip this validity check for $i = 0$ in line-6 in Algorithm 4, then $seed[0]$ will be revealed. Henceforth all \tilde{Q}_i 's would have been revealed. Therefore, one may want to protect this checking at any cost. In fact, the need to protect this validity check was previously noted by Oder et al. [26] for different post-quantum schemes (e.g. Kyber). Nevertheless, the data in \mathbf{d} and *Reference Tree* \mathbf{x} can be corrupted by skipping the storing instruction and forcing the data not to change.

Skip the Store Instruction in Algorithm 5 to Corrupt \mathbf{x} : All the nodes of the *Reference Tree* \mathbf{x} are initialized by zero. If we can skip the store instruction $\mathbf{x}[i] = \mathbf{x}[2i+1] \vee \mathbf{x}[2i+2]$ for any i , then value of $\mathbf{x}[i]$ will remain zero. However, if the value of $\mathbf{x}[i]$ is supposed to be 1 in the non-faulted case, then $\mathbf{x}[i]$ will be modified after injecting this store instruction fault. As we discussed earlier, we can find the information of the secret matrix if the fault changes the value of $\mathbf{x}[i]$ from 1 to 0 and $\mathbf{x}[0]$ remains 1. A similar instruction skipping attack has been shown in [28].

Stuck-at-Zero Fault Model to Corrupt \mathbf{x} : A possible attack avenue is exploiting effective faults in the stuck-at-zero model, where an attacker can try to alter the i -th intermediate value $\mathbf{x}[i]$ to a particular known value, e.g., to zero using stuck-at-zero fault [13, 17, 20] using voltage glitches or electromagnetic attacks. The effect of this fault is equivalent to the above “store instruction skipping” fault. So, this fault will allow us to find the secret matrix.

Rowhammer Attack Model to Corrupt \mathbf{x} : Rowhammer [31] is a hardware bug identified in DRAMS (dynamic random access memory), where repeated row activations can cause bitflips in adjacent rows. This can also be a possible attack where bitflips ($1 \rightarrow 0$) can be employed to corrupt the data $\mathbf{x}[i]$. Recently, such an attack on Kyber using rowhammer has been shown in [24].

As we discussed, any one of the above fault models can generate effective faulted signatures. However, the definition of successful fault always depends on the fault model. For example, if we work on the first fault model, *i.e.*, “Skip the validity check condition in Algorithm 4”, then the successful-fault will be: successfully skipped the checking condition “ $\mathbf{x}[i] = 0$ and $\mathbf{x}[Parent(i)] = 1$ ” for a known i . But, if we work on the second fault model, then the successful fault will be: successfully skipped the store instruction “ $\mathbf{x}[i] = \mathbf{x}[2i+1] \vee \mathbf{x}[2i+2]$ ” for a known i .

From now on, we will discuss the second fault model to inject a fault, *i.e.*, we inject a fault to skip the i -th store instruction $Ins(i) : \mathbf{x}[i] = \mathbf{x}[2i+1] \vee \mathbf{x}[2i+2]$ ”

for a fixed known i . Since all the values of the *Reference Tree* \mathbf{x} are initialized by zero, therefore for each successful fault, the value of $\mathbf{x}[i]$ will always be zero, where the position of fault location $\mathbf{x}[i]$ is known to the attacker. In the practical setup of this store instruction skip fault model, the following cases may arise:

- Successfully skipped the instruction $Ins(i)$, for the known i and outputs the signature we call it a successful faulted signature. This fault could be an effective or ineffective fault.
- Could not skip the instruction $Ins(i)$, for the known fixed i . In this case, we call the output signature an unsuccessful faulted signature.

In a physical device, faults can be induced with varying success rates. Even if there is a successful fault, the resulting faulty signature may or may not provide information about the secret key, as we observed earlier. A successful fault is called “effective” if it reveals secret key information and “ineffective” if it does not. More explicitly, we will say that a successful fault is effective if the fault changes the value of $\mathbf{x}[i]$ from 1 to 0, but the value of the root node $\mathbf{x}[0]$ remains unchanged, (i.e., 1). Otherwise, the fault will be ineffective. We must identify the effective faulted signature from the received signature to find the errorless secret matrix. In the next Sect. 3.4, we will discuss the effective faulted signature detection method.

3.4 Effective Fault Detection

Let $\tau' = (salt, cmt, \mathbf{TreeNode}', \mathbf{rsp})$ be the received signature corresponding to the message m . We need to detect if the signature is generated from an “effective” or “ineffective” fault. The injected fault only affects the *Reference Tree* \mathbf{x} , so the signature components $salt$, cmt and \mathbf{rsp} remain the same with the corresponding non-faulted signature components. We can compute the fixed digest vector \mathbf{d} corresponding to the signature τ' from cmt . From the fixed digest vector \mathbf{d} , we can compute the vector \mathbf{f} and the *Reference Tree* \mathbf{x} for the non-faulted case. However, we can compute the successful faulted *Reference Tree* \mathbf{x}' from the *Reference Tree* \mathbf{x} by assigning the value of $\mathbf{x}'[i] = 0$ and updating the ancestors of $\mathbf{x}'[i]$ accordingly, i.e., \mathbf{x}' should be the *Reference Tree* if the instruction $Ins(i)$ is skipped. Then, we will distinguish the “effective” faulted signature and “ineffective” faulted signature with the following process:

- **Step-1:** First, we will check whether $\mathbf{x}[i] = 0$ or not. If $\mathbf{x}[i] = 0$, then this is already a case of “ineffective fault”, and we reject the signature. Otherwise, we will go to the next step.
- **Step-2:** Next, we will check whether $\mathbf{x}'[0] = 0$ or not. If $\mathbf{x}'[0] = 0$, then this is a case of “ineffective fault”, and we reject the signature. Otherwise, from the *Reference Tree* \mathbf{x}' , we compute the size of successfully faulted $\mathbf{TreeNode}'$, say Δ_{exp} and the size of received $\mathbf{TreeNode}'$ say Δ_{rec} . We compare the values Δ_{exp} with Δ_{rec} .
- **Step-3:** If $\Delta_{rec} \neq \Delta_{exp}$, then the fault is unsuccessful, and we reject the signature. Otherwise, using $salt$, $\mathbf{TreeNode}'$ and \mathbf{x}' we compute all the \tilde{Q}_j

where $\mathbf{d}[j] = 0$. We apply the verification using these \tilde{Q}_j 's and \mathbf{rsp} . If the verification is successful, then we take the received signature as an effective faulted signature. Otherwise, we reject the signature.

Note that, in **Step-3** of the above process, $\mathbf{x}[i]$ is changed from $1 \rightarrow 0$ and $\mathbf{x}[0] = 1$, but we still consider it as “unsuccessful” fault. This is because we want our fault detection method to detect whether our fault has been successfully injected exactly at the i -th location or not. If $\mathbf{x}[i]$ changed from $1 \rightarrow 0$, then there are two cases.

- *Case-1*: fault was successfully injected at i -th location.
- *Case-2*: fault was injected at a j -th location for $j \neq i$ and it has changed $\mathbf{x}[i]$.

We only consider *Case-1* as “successful-fault”, but not *Case-2* as the fault is not injected at the i -th location in that case. In this procedure, we can detect that the faulted signature that is generated by successfully skipping the instruction $Ins(i)$ and that leaks the information about the secret matrix. Note that the targeted faulted location i is arbitrary but known to the attacker. For simplicity, we will fix the targeted fault position i . We will check whether this fixed i -th store instruction $Ins(i)$ skipped and that leaks the information about the secret matrix or not. If this detection method passes, then we will use this signature. Otherwise, we will again query for another signature.

3.5 Attack Template

In this section, first, we will describe how to obtain the secret monomial matrices from an effective faulted signature $\tau = (\mathit{salt}, \mathit{cmt}, \mathit{TreeNode}, \mathit{rsp})$ in Algorithm 6. Let $\mathbf{x}[i]$ be the node in *Reference Tree* with height h , and $L_{\mathbf{x}[i]}$ be the set of all leaf nodes of the subtree rooted at $\mathbf{x}[i]$. We only need the leaf nodes from $L_{\mathbf{x}[i]}$ that coincide with the first t (the length of the digest \mathbf{d}) many leaf nodes of the full *Reference Tree*. Without loss of generality, assume that there are v many such leaves, and let the set of indices of these leaves be $I_{\text{leaf}}^{(i)} = \{j_1, j_2, \dots, j_v\}$. From this effective faulted signature, all the secret matrices $\mathbf{Q}_{\mathbf{d}[j-2l+1]}^T$ will be recovered with Algorithm 6, where $j \in I_{\text{leaf}}^{(i)}$ and $\mathbf{d}[j-2l+1] \neq 0$.

Here, $\mathit{SeedTreeUpdate}$ function takes the $\mathit{TreeNode}$, salt and digest \mathbf{d} and generates all the ephemeral seeds assuming the modified *Reference Tree* after effective fault. Since $\mathit{seed}[i]$ is revealed in $\mathit{TreeNode}$ after effective fault, we can say that $\mathit{ESEED}[j-2l+1]$ for all $j \in I_{\text{leaf}}^{(i)}$ are revealed.

In this attack model, we are able to get into the victim’s device and introduce the fault that causes it to bypass the $Ins(i)$ instruction. The $\mathit{LESS_KeyGen}$ (Algorithm 1) is a one-time operation from where the secret key $\mathit{SK} = (\mathit{MSSED}, \mathit{gseed})$ and public key $\mathit{PK} = (\mathit{gseed}, \mathbf{G}_1, \dots, \mathbf{G}_{s-1})$ are generated. But with this private key SK , the $\mathit{LESS_Sign}$ (Algorithm 2) can execute more than once. We follow the following subsequent actions to find the secret monomial matrices:

Algorithm 6. Recover_Secret_Matrices(τ, PK)

Input: Signature $\tau = (\text{salt}, \text{cmt}, \text{TreeNode}, \text{rsp})$, public key $PK = (\text{gseed}, \mathbf{G}_1, \dots, \mathbf{G}_{s-1})$.

Output: The columns of secret matrices $\mathbf{Q}_{d[j-2l+1]}^T$, where $j \in I_{\text{leaf}}^{(i)}$ and $d[j-2l+1] \neq 0$.

```

1:  $\mathbf{d} \leftarrow \text{CSPRNG}(\text{cmt}, \mathbb{S}_{t,w})$ 
2:  $\text{seed} \leftarrow \text{SeedTreeUpdate}(\text{TreeNode}, \text{salt}, \mathbf{d})$ 
3:  $\text{ESEED} \leftarrow \text{Leaf nodes of seed corresponding to seed}[i]$ 
4:  $\mathbf{G}_0 \leftarrow \text{CSPRNG}(\text{gseed}, \mathbb{S}_{\text{REF}})$ 
5: for  $r = 1; r \leq v; r = r + 1$  do
6:   if  $d[j_r - 2l + 1] \neq 0$  and  $\mathbf{Q}_{d[j_r - 2l + 1]}$  is not recovered then
7:      $\tilde{\mathbf{Q}}_{j_r - 2l + 1} \leftarrow \text{CSPRNG}(\text{ESEED}[j_r - 2l + 1], M_n(q))$ 
8:      $(\bar{\mathbf{Q}}_{j_r - 2l + 1}, \bar{\mathbf{V}}_{j_r - 2l + 1}) \leftarrow \text{PrepareDigestInput}(\mathbf{G}_0, \tilde{\mathbf{Q}}_{j_r - 2l + 1})$ 
9:      $\mathbf{Q}^* = \mathbf{Q}_{d[j_r - 2l + 1]}^T \bar{\mathbf{Q}}_{j_r - 2l + 1} \leftarrow \text{ExpandToMonomAction}(\text{rsp})$ 
10:    Compute  $\mathbf{Q}_{d[j_r - 2l + 1]}^T$  from  $(\mathbf{Q}^*, \bar{\mathbf{Q}}_{j_r - 2l + 1})$  ▷ following Section 3.1

```

- **Step-1:** We generate a message, signature pair (m, τ) from the victim device.
- **Step-2:** After receiving the pair (m, τ) , we will determine whether or not τ is an effective faulted signature. Go back to **Step-1** if the signature is not effective. If yes, then go to **Step-3**.
- **Step-3:** Using this signature τ , we will run the `Recover_Secret_Matrices` algorithm (Algorithm 6) to determine the hidden monomial matrices.
- **Step-4:** Next, we will calculate whether or not the whole secret monomial matrices were obtained. We terminate the process if the secret matrices are recovered. Otherwise, we repeat the same procedure to obtain the remaining non-recovered columns.

3.6 Secret Recovery from Single Fault

In this section, we calculate the expected number of secret monomials recovered from one effective faulted signature where the fault is injected at a node $\mathbf{x}[i]$ ($0 \leq i \leq 4l - 2$). Now, if there are m many non-zero leaves with distinct values in the subtree rooted at $\mathbf{x}[i]$, then we will get exactly m many pairs of the form $(\tilde{\mathbf{Q}}_j, \mathbf{Q}_{d[j]}^T \tilde{\mathbf{Q}}_j)$ i.e. we recover m many secret monomials. In this section, first, we will estimate the value of m .

Suppose $L_{\mathbf{x}[i]}$ the set of leaf nodes in the subtree rooted at $\mathbf{x}[i]$ and let $|L_{\mathbf{x}[i]}| = \ell$. Let W be the random variable representing the number of leaf nodes in $L_{\mathbf{x}[i]}$ with non-zero value. X be a random variable that represents the number of distinct non-zero values of the leaf nodes in $L_{\mathbf{x}[i]}$. Then for any $0 \leq m \leq s - 1$, we have

$$\Pr[X = m] = \sum_{r=m}^w \Pr[X = m \mid W = r] \cdot \Pr[W = r]$$

Where w is the weight of \mathbf{d} and therefore $L_{\mathbf{x}[i]}$ can only have at most w many non-zero valued leaf nodes. First, we will calculate $\Pr[X = m \mid W = r]$, which

is the probability that r many non-zero leaves take exactly m many distinct values. These m distinct values can be chosen from $(s - 1)$ possible values in $\binom{s-1}{m}$ ways. Now, we have to assign all these m values to the r many leaf nodes. We first partition the r locations into m many non-empty subsets, which can be done in $S(r, m)$ many ways. This $S(r, m)$ is a *Stirling number of the second kind* [32]. Now, each of the m many subsets can be assigned a unique non-zero value, which can be done in $m!$ ways. So, the r many leaf nodes can be assigned m distinct value in $m! \binom{s-1}{m} S(r, m)$ ways. Therefore

$$\Pr[X = m \mid W = r] = \frac{m! \binom{s-1}{m} S(r, m)}{(s-1)^r}$$

Now, \mathbf{d} has weight w and $\Pr[W = r]$ is the probability that the ℓ many locations of \mathbf{d} corresponding to the leaf nodes in $L_{\mathbf{x}[i]}$ has exactly r many non-zero values and the last $t - \ell$ many locations has $w - r$ many non-zero values. Therefore,

$$\Pr[W = r] = \frac{\binom{\ell}{r} \binom{t-\ell}{w-r}}{\binom{t}{w}}$$

Now we can calculate $\Pr[X = m]$ for all $0 \leq m \leq s - 1$. However, we are interested in finding the expected number of secret monomials with one single fault, which is the expectation of the random variable X .

$$\begin{aligned} \mathbb{E}[X] &= \sum_{m=1}^{s-1} m \cdot \Pr[X = m] \\ &= \sum_{m=1}^{s-1} m \left(\sum_{r=m}^w \frac{m! \binom{s-1}{m} S(r, m)}{(s-1)^r} \cdot \frac{\binom{\ell}{r} \binom{t-\ell}{w-r}}{\binom{t}{w}} \right) \end{aligned}$$

With only one single faulted signature the expected number of secret monomials that we recover is $\mathbb{E}[X]$ but the total number of secret monomials is $(s - 1)$. Therefore, we need multiple faulted signatures to recover all the secret monomials.

4 Extending Our Attack to CROSS

CROSS uses \mathbb{E}^n a commutative group isomorphic to $(\mathbb{F}_z^n, +)$, where n, z are parameters of the signature and G is a subgroup of \mathbb{E}^n . Here \mathbf{e} is a long-term secret vector which is used to generate signatures. Therefore, the attacker can generate multiple valid signatures using the secret \mathbf{e} information. Similar to the attack on LESS, the target here is to find the information of the secret vector \mathbf{e} . In the Algorithm 7, we can observe that if we have information of one single pair $(\mathbf{e}'^{(i)}, f^{(i)} = (\mathbf{y}^{(i)}, \sigma^{(i)}, c_1^{(i)}))$, then we can compute the secret \mathbf{e} by $\mathbf{e} = \sigma^{(i)}(\mathbf{e}'^{(i)})$. Therefore, we aim to find one such pair corresponding to any i for full key recovery.

Algorithm 7. CROSS_Sign (Msg, e)

Input: Secret key $e \in G$ and message Msg where $G \subset \mathbb{E}^n$, $\mathbf{H} \in \mathbb{F}_p^{(n-k) \times n}$ are public key satisfying $s = e\mathbf{H}^T$

Output: Signature $\tau = \{Salt, c_0, c_1, h, SeedPath, MerkleProofs, \{f^{(i)}\}_{i \notin J}\}$

```

1: Sample  $MSeed \xleftarrow{\$} \{0, 1\}^\lambda$ ,  $Salt \xleftarrow{\$} \{0, 1\}^{2\lambda}$ 
2: Generate  $Seed = SeedTree(MSeed, Salt)$ 
3:  $ESEED[1], \dots, ESEED[t] =$  Leaf nodes of  $Seed$ 
4: for  $i = 1, i \leq t, i = i + 1$  do
5:   Sample  $(Seed^{(u')}, Seed^{(v')}) \xleftarrow{ESEED[i]} \{0, 1\}^{2\lambda}$ 
6:   Sample  $u'^{(i)} \xleftarrow{Seed^{(u')}} \mathbb{F}_p^n, e'^{(i)} \xleftarrow{Seed^{(e')}} G$ 
7:   Compute  $\sigma^{(i)} \in G$  such that  $\sigma^{(i)}(e'^{(i)}) = e$ 
8:   Set  $u^{(i)} = \sigma^{(i)}(u'^{(i)})$ 
9:   Compute  $\tilde{s}^{(i)} = u^{(i)}\mathbf{H}^T$ 
10:  Set  $c_0^{(i)} = Hash(\tilde{s}^{(i)}, \sigma^{(i)}, Salt, i)$ 
11:  Set  $c_1^{(i)} = Hash(u'^{(i)}, e'^{(i)}, Salt, i)$ 
12: Set  $\mathcal{T} = MerkleTree(c_0^{(1)}, \dots, c_0^{(t)})$ 
13: Compute  $c_0 = \mathcal{T}.Root()$ 
14: Compute  $c_1 = Hash(c_1^{(1)}, \dots, c_1^{(t)})$ 
15: Generate  $(\beta^{(1)}, \dots, \beta^{(t)}) = GenCh_1(c_0, c_1, Msg, Salt)$ 
16: for  $i = 1, i \leq t, i = i + 1$  do
17:   Compute  $y^{(i)} = u'^{(i)} + \beta^{(i)}e'^{(i)}$ 
18:   Compute  $h^{(i)} = Hash(y^{(i)})$ 
19: Compute  $h = Hash(h^{(1)}, \dots, h^{(t)})$ 
20: Generate  $(b[1], \dots, b[t]) = GenCh_2(c_0, c_1, \beta^{(1)}, \dots, \beta^{(t)}, h, Msg, Salt)$ 
21: Set  $J = \{i : b[i] = 1\}$ 
22: Set  $SeedPath = publish\_seeds(MSeed, Salt, J)$ 
23: for  $i \notin J$  do
24:    $f^{(i)} := (y^{(i)}, \sigma^{(i)}, c_1^{(i)})$ 
25: Compute  $MerkleProofs = \mathcal{T}.Proofs(\{1, \dots, t\} \setminus J)$ 
26: Return  $\tau = \{Salt, c_0, c_1, h, SeedPath, MerkleProofs, \{f^{(i)}\}_{i \notin J}\}$ 

```

In Algorithm 7, the function `publish_seeds` (line 22) works equivalent to the function `SeedTreePaths` (Algorithm 4) used in LESS signature. Using the digest vector \mathbf{b} , the function `publish_seeds` first creates a *Reference Tree* say \mathbf{y} in a bottom-up approach like LESS signature. The only difference in this *Reference Tree* \mathbf{y} is that the flag of the published seed is defined as 1 and unpublished seed notation as 0, whereas in LESS, the authors define the opposite. However, Both use equivalent concepts. Like LESS, we require the modification of any node of the *Reference Tree* from 1(flag of the unpublished seed) \rightarrow 0(flag of the published seed) to get an effective faulted signature. Therefore, we need the modification from 0 \rightarrow 1 to get an effective faulted signature. We can detect the effective faulted signature here using a similar technique that we used in Sect. 3.4 to detect effective fault for LESS signature.

Let us assume that we apply fault injection to the CROSS signature of a victim's device such that the value of $\mathbf{y}[i]$ has been

Algorithm 8. Recover_Secret_CROSS (τ, PK)

Input: $\tau = \left\{ Salt, c_0, c_1, h, \mathbf{SeedPath}, \mathbf{MerkleProofs}, \left\{ (\mathbf{y}^{(i)}, \sigma^{(i)}, c_1^{(i)}) \right\}_{i \notin J} \right\}$

Output: The secret vector \mathbf{e} .

- 1: Generate $(\beta^{(1)}, \dots, \beta^{(t)}) = \text{GenCh}_1(c_0, c_1, \text{Msg}, \text{Salt})$
- 2: Generate $(\mathbf{b}[1], \dots, \mathbf{b}[t]) = \text{GenCh}_2(c_0, c_1, \beta^{(1)}, \dots, \beta^{(t)}, h, \text{Msg}, \text{Salt})$
- 3: Set $J = \{i : \mathbf{b}[i] = 1\}$
- 4: $\mathbf{ESEED}[j_1 - 2l], \dots, \mathbf{ESEED}[j_v - 2l] \leftarrow \text{SeedTreeUpdate}(\mathbf{seed}[i], \text{Salt}, \mathbf{b})$
- 5: **for** $i = j_1 - 2l, \dots, j_v - 2l$: **do**
- 6: **if** $i \notin J$ **then**
- 7: Sample $(\text{Seed}^{(u')}, \text{Seed}^{(v')}) \xleftarrow{\mathbf{ESEED}[i]} \{0, 1\}^{2\lambda}$
- 8: Sample $\mathbf{e}'^{(i)} \xleftarrow{\text{Seed}^{(e')}} G$
- 9: Compute $\mathbf{e} = \sigma^{(i)}(\mathbf{e}'^{(i)})$
- 10: **return** \mathbf{e}

changed from $0 \rightarrow 1$. Consider an effective faulted signature as $\tau = \left\{ Salt, c_0, c_1, h, \mathbf{SeedPath}, \mathbf{MerkleProofs}, \left\{ f^{(i)} \right\}_{i \notin J} \right\}$. Since τ is an effective-faulted signature, therefore we will get the seed $\mathbf{seed}[i]$. All the leaf nodes of the subtree say $L_{\mathbf{x}[i]} = \{\mathbf{x}[j_1], \dots, \mathbf{x}[j_v]\}$ rooted as $\mathbf{seed}[i]$ can be computed from $\mathbf{seed}[i]$. i.e., $\mathbf{ESEED}[j_1 - 2l], \dots, \mathbf{ESEED}[j_v - 2l]$ will be the corresponding leaf ephemeral seeds. Now, we will find the secret key \mathbf{e} using the Algorithm 8. The function SeedTreeUpdate works the same way we defined it in Sect. 3.5.

5 Simulation Result

In this section, we discuss the simulation procedure of our fault attack on LESS and CROSS signatures; i.e., we apply our fault assumption inside the LESS and CROSS signature algorithms to imitate the corresponding practical attack scenario. The simulation code is available at GitHub².

In the previous Sects. 3.6 and 4, we have analyzed the effect of modification of the values $\mathbf{x}[i]$ (for LESS) and $\mathbf{y}[i]$ (for CROSS) to 0 and 1 respectively. For LESS and CROSS, this can be achieved by stuck at zero/stuck at one or instruction skip fault. So, in the simulation code, we have assumed the values 0 and 1 of the nodes $\mathbf{x}[i]$ and $\mathbf{y}[i]$ respectively. After receiving this faulted signature τ , we compute the corresponding secrets of CROSS (secret \mathbf{e}) and LESS (secret monomial matrices) with the help of the respective algorithms Algorithm 6 and Algorithm 8.

Note that the attack is valid if we target any $\mathbf{x}[i]$ ($\mathbf{y}[i]$) for fault injection in LESS (CROSS) signature, where i is an arbitrary but fixed location. But in our simulation code we have fixed the location as $i = 1$. One may change this location and the simulation code accordingly. However, in that case the results

² https://github.com/s-adhikary/zkfault_simulation.

in Table 2 would change according to our result in Sect. 3.6. We provide the simulation results for all the versions of LESS and CROSS in Table 2. We have run the simulation code multiple times to recover all secrets with (multiple) faulted signatures. We take the average of the number of faulted signatures required to recover all secrets, which we denote with N_{avg} . We have also included the average number of secrets recovered from one single fault ($\mathbb{E}[X]$) in the table.

Table 2. Simulation result of full secret monomial matrices recovery of LESS and CROSS signature [7,34].

Scheme	Security Level	Parameter Set	Optim. Corner	Number of Secrets	$\mathbb{E}[X]$	N_{avg}
LESS [7]	1	LESS-1b	-	1	1	1
		LESS-1i	-	3	2.91	1.05
		LESS-1s	-	7	5.55	2.09
	3	LESS-3b	-	1	1	1
		LESS-3s	-	2	2	1
	5	LESS-5b	-	1	1	1
LESS-5s		-	2	2	1	
CROSS [34]	1, 3, & 5	CROSS-R-SDP	fast/small	1	1	1
		CROSS-R-SDP(G)	fast/small	1	1	1

Our analysis is based on the fact that each time we query the faulted signature oracle, we get an effectively faulted signature. However, in a practical fault attack, this is not the case. In the real world, there is a probability that an injected fault is successful, say p_1 , and also there is a probability that a successfully injected fault is effective, say p_0 . Then

$$\begin{aligned}
 & \Pr[\text{effective fault} \wedge \text{successful fault}] \\
 &= \Pr[\text{effective fault} \mid \text{successful fault}] \cdot \Pr[\text{successful fault}] \quad (5) \\
 &= p_0 p_1
 \end{aligned}$$

Let us consider $p = p_0 p_1$. Therefore, in a practical scenario to get one faulted signature, the approximate number of queries to the faulted signature oracle needed would be $N_{\text{trial}} = \frac{1}{p}$. Moreover, to get N_{avg} many faulted signatures, we need $N_{\text{total}} = \frac{N_{\text{avg}}}{p}$ many queries. For example if we consider $p = 0.01$, then $N_{\text{trial}} = 100$ and consequently, $N_{\text{total}} = 100 \cdot N_{\text{avg}}$.

6 Countermeasures

In the previous section, we have seen that the primary attack surface *Reference Tree* x is initialized by 0. If we inject fault to skip store instruction

line-5 of Algorithm 5 i.e. $\mathbf{x}[i] = \mathbf{x}[2i+1] \wedge \mathbf{x}[2i+2]$, then $\mathbf{x}[i]$ does not change the value and stays 0. Hence, one may suggest initializing the *Reference Tree* with all 1. The instruction skip fault does not work in this case, but we can apply bit-flip fault or stuck-at-zero faults and apply the same attack analysis. Since many practical fault attacks are applicable, countermeasures against one type of fault may not serve our purpose. Therefore, first, we must identify the main reason for the existence of the attack vector.

After the digest computation in Algorithm 2, the values of vector \mathbf{d} are checked twice. First, by checking whether the value of each $\mathbf{d}[i]$ is zero or not, they published the component *TreeNode*. Completing this procedure, again, each $\mathbf{d}[i]$ is checked to publish the component *rsp*. Therefore, if an attacker injects a fault at the time of computing *TreeNode* and somehow succeeds in disclosing the seed $\mathbf{ESEED}[i]$ without altering the vector \mathbf{d} , then the information about the secret matrix $\mathbf{Q}_{\mathbf{d}[i]}^T$ is susceptible to leakage. To mitigate potential attacks, we must publish either the response $\tilde{\mathbf{Q}}_i$ or $\mathbf{Q}_{\mathbf{d}[i]}^T \bar{\mathbf{Q}}_i$ after a single verification of the value $\mathbf{d}[i]$.

In the following sections, we will offer concise explanations for two countermeasures incorporated within the LESS scheme that protect the scheme up to one fault.

6.1 Countermeasure with Larger Signature Size

The most straightforward countermeasure would be not using the tree construction at all. In this version, the preparation of digest \mathbf{d} is the same as Algorithm 2. After the digest preparation, for each $0 \leq i < t$ we only check the value of $\mathbf{d}[i]$, and set

$$rsp[i] = \begin{cases} \tilde{\mathbf{Q}}_i & \text{if } \mathbf{d}[i] = 0 \\ \mathbf{Q}_{\mathbf{d}[i]}^T \bar{\mathbf{Q}}_i & \text{otherwise} \end{cases} .$$

Then, we cannot get both $\tilde{\mathbf{Q}}_i$ and $\mathbf{Q}_{\mathbf{d}[i]}^T \bar{\mathbf{Q}}_i$ for any i and the attack can be prevented. However, in this case, the size of the signature will be $|cmt| + wk(\lceil \log n \rceil + \lceil \log(q-1) \rceil) + (t-w)\lambda$. This signature size will be larger than the submitted version of LESS [35].

6.2 Countermeasure with Same Small Signature Size

Here, we introduce another countermeasure that will keep the signature size the same as the submitted version of LESS [35]. In this countermeasure, our main target is that after computation of the *Reference Tree* \mathbf{x} , we check the *Reference Tree* only once to compute the response for the signature. Note that, according to the construction of the *Reference Tree* \mathbf{x} , the path from a leaf node to the root node should be of form $0^y 1^z$ because if the value of any node on this path is 1, then all the ancestors of that node will be 1.

After the preparation of the *Reference Tree* \mathbf{x} , we modify the signature generation method. First, observe that for any leaf node $\mathbf{x}[2l-1+i]$ of the *Reference*

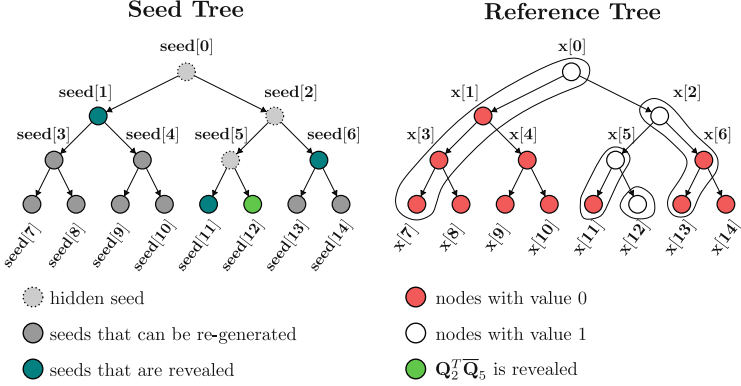


Fig. 5. Example for extraction of response \mathbf{rsp} , $\mathbf{TreeNode}$ according to Algorithm 9

Tree, let the path to the root be $\mathbf{x}[i_0]\mathbf{x}[i_1]\cdots\mathbf{x}[i_p]$, where $p = \lceil \log_2(l) \rceil + 1$ is the height of the *Reference Tree*. Here, $i_0 = 2l - 1 + i$ and $\mathbf{x}[i_j]$ is the ancestor of $\mathbf{x}[2l - 1 + i]$ at the height j , this means that $\mathbf{x}[i_p]$ is the root. The following is the signature generation process:

- **Step-1:** We start from the leftmost leaf node.
- **Step-2:** check the path $\mathbf{x}[i_0]\mathbf{x}[i_1]\cdots\mathbf{x}[i_p]$
- **Step-3:** If $\mathbf{x}[i_j] = 1$ for $0 \leq j \leq p$, then we store $\mathbf{Q}_{d[i]}^T \bar{\mathbf{Q}}_i$ in \mathbf{rsp} and select the next leaf node as $\mathbf{x}[2l + i]$ and goto **Step-2**, else go to **Step-4**.
- **Step-4:** We find $\mathbf{x}[i_h]$, which is the highest ancestor of $\mathbf{x}[i_0]$ with the value zero.
- **Step-5:** Since $\mathbf{x}[i_h] = 0$, all the 2^h leaf nodes of the subtree rooted $\mathbf{x}[i_h]$ must be zero. We store the seed $\mathbf{seed}[i_h]$ in $\mathbf{TreeNode}$ and we select the next leaf node as $\mathbf{x}[2l - 1 + i + 2^h]$ and go to **Step-2**. If no more leaf nodes are left, then we stop.
- **Step-6:** return the pair \mathbf{rsp} and $\mathbf{TreeNode}$.

The digest \mathbf{d} and the *Reference Tree* are prepared in the same process here as it is prepared in Algorithm 2. Only the vectors \mathbf{rsp} , $\mathbf{TreeNode}$ are prepared using Algorithm 9. At the end, $(\mathbf{cmt}, \mathbf{salt}, \mathbf{rsp}, \mathbf{TreeNode})$ is generated as the signature.

Example 4. Given a fixed signature digest vector represented as $\mathbf{d} = (0, 0, 0, 0, 0, 2, 0, 0)$. First, we construct the *Reference Tree* \mathbf{x} , which is illustrated in Fig. 5. Begin by checking leaf nodes from the left side. First, we take the leftmost leaf node $\mathbf{x}[7]$. The path from $\mathbf{x}[7]$ to root is $\mathbf{x}[i_0]\mathbf{x}[i_1]\mathbf{x}[i_2]\mathbf{x}[i_3]=\mathbf{x}[7]\mathbf{x}[3]\mathbf{x}[1]\mathbf{x}[0]$, and it is valued 0001. In Fig. 5, we can see that the height of the last ancestor valued 0 is $h' = 3$, and the node is $\mathbf{x}[1]$. We store $\mathbf{seed}[1]$ in response $\mathbf{TreeNode}$ and select the next leaf node as $\mathbf{x}[7 + 2^{h'-1}] = \mathbf{x}[11]$. Final response will be calculated as $\mathbf{rsp} = (\mathbf{Q}_2^T \bar{\mathbf{Q}}_{12})$ and $\mathbf{TreeNode} = (\mathbf{seed}[1], \mathbf{seed}[11], \mathbf{seed}[6])$.

Algorithm 9. LESS_Gen_rsp_update

Input: The fixed weight digest vector \mathbf{d} , The secret monomial matrices $\mathbf{Q}_i, \forall i \in \mathbb{Z}_s$, where $\mathbf{Q}_0 = \mathbf{I}_n$, The *Seed Tree* seed , and the partial monomial matrices $\overline{\mathbf{Q}}_j, \forall j \in \mathbb{Z}_t$.

Output: The response rsp and *TreeNode*

```

1: for  $i = 0; i < t; i = i + 1$  do
2:   if  $\mathbf{d}[i] = 0$  then
3:      $\mathbf{f}[i] = 0$ 
4:   else
5:      $\mathbf{f}[i] = 1$ 
6:   for  $i = 0; i < 4l; i = i + 1$  do
7:      $\mathbf{x}[i] = 0$ 
8:    $\mathbf{x} \leftarrow \text{compute\_seeds\_to\_publish}(\mathbf{f}, \mathbf{x})$ 
9:    $i = 0, j = 0, j' = 0$ 
10:  while  $i < t$  do
11:     $c = 2l - 1 + i, h = 0, h' = 0$ 
12:    while  $\text{Parent}(c) \neq 0$  do
13:      if  $\mathbf{x}[c] = 0$  then
14:         $c' = c, h' = h + 1$ 
15:         $c = \text{Parent}(c), h = h + 1$ 
16:      if  $h' = 0$  then
17:         $\mathit{rsp}[j'] = \text{CompressMono}(\mathbf{Q}_{\mathbf{d}[i]}^T \overline{\mathbf{Q}}_i)$ 
18:         $i = i + 1, j' = j' + 1$ 
19:      else
20:         $\mathit{TreeNode}[j] = \mathit{seed}[c']$ 
21:         $i = i + 2^{h'-1}, j = j + 1$ 
21: Return  $\mathit{rsp}, \mathit{TreeNode}$ 

```

Suppose we inject a fault at the node $\mathbf{x}[i]$ and alter its value from 1 to 0. Then some of its ancestors may change. Let $\mathbf{x}[i_1], \mathbf{x}[i_2], \dots, \mathbf{x}[i_h]$ be the list of all ancestors of $\mathbf{x}[i]$, where $\mathbf{x}[i_j]$ is ancestor of $\mathbf{x}[i_{j-1}]$ for all $j \in [2, h]$ and $\mathbf{x}[i_h]$ is the root. Suppose $\mathbf{x}[i_y]$ is the highest ancestor in the list to have the value zero. Now, consider the leftmost leaf node $\mathbf{x}[r]$ of the subtree rooted at $\mathbf{x}[i_y]$, then $\mathbf{x}[i_y]$ is the highest node with value zero in the path from $\mathbf{x}[r]$ to root. Hence, according to Algorithm 9, $\mathit{seed}[i_y]$ is appended to *TreeNode* and all the leaf nodes in the subtree rooted at $\mathbf{x}[i_y]$ are skipped.

Observe that the fault at $\mathbf{x}[i]$ only affects the subtree rooted at $\mathbf{x}[i_y]$, the rest of the *Reference Tree* is unchanged. The subtree rooted at $\mathbf{x}[i_y]$ is skipped after revealing $\mathit{seed}[i_y]$, and $\mathit{seed}[i_y]$ can only be used to generate the ephemeral seeds that do not have any information about the secret monomial matrices. Therefore, the attack will not be possible with just one fault.

We only change the attack surface part to protect the LESS scheme against our attack. The attack surface of the CROSS signature scheme is similar to LESS. We can use the proposed countermeasure for CROSS also. We only need to modify the update method of rsp and *TreeNode* according to the CROSS signing algorithm.

Cost of the Countermeasure. Here we will compare the cost analysis of our proposed countermeasure with the original LESS implementation (Algorithm 2). The *Reference Tree* generation process in our proposed method is the same as the original LESS proposal. We have only changed the **TreeNode** and **rsp** generation process but result is same in both cases *i.e.*, for a particular *Seed Tree*, *Reference Tree* pair, our method and original LESS implementation, both generate the same **TreeNode** and **rsp**. First of all, we consider the following computation costs:

- C_{check} : cost of any condition checking
- C_{mono} : cost of computation of a monomial multiplication followed by a **CompressMono** function and storing the result
- C_{seed} : cost of storing seeds from *Seed Tree* condition checking

Also, we fix a *Seed Tree* and a *Reference Tree* and we assume that r many seeds from *Seed Tree* are to be stored in **TreeNode**. Assume that the total number of nodes in the *Reference Tree* is N .

Cost of LESS Original Implementation (Algorithm 2): As we can see in Algorithm 2, the **TreeNode** is generated using Algorithm 4. We are going to ignore the cost of `compute_seeds_to_publish` function (Algorithm 5), as it has also been used in our countermeasure. For each node in the *Reference Tree*, Algorithm 4 checks the node and its parent node which takes $2N \cdot C_{\text{check}}$ computations. As we have assumed earlier there are r many seeds which are to be stored in **TreeNode**, which takes $r \cdot C_{\text{seed}}$ computations. After that, Algorithm 2 checks each value of the vector \mathbf{d} which takes $t \cdot C_{\text{check}}$ computations and computes the monomial multiplication and calls the **CompressMono** function for each $\mathbf{d}[i] \neq 0$, which takes $w \cdot C_{\text{mono}}$ computations. Therefore the total cost is $(2N + t) \cdot C_{\text{check}} + r \cdot C_{\text{seed}} + w \cdot C_{\text{mono}}$.

Cost of Our Proposed Method (Algorithm 9): In each iteration of the while loop, we first check the full path from the leaf node to the root, this takes $\log_2 N \cdot C_{\text{check}}$ computations. In each iteration, the algorithm can either update **TreeNode** or update **rsp** *i.e.*, the total number of iterations in the while loop is $(r + w)$. The condition checking takes total $(r + w) \cdot \log_2 N \cdot C_{\text{check}}$ computations. Now the **rsp** is updated for w many iterations, which takes total $w \cdot C_{\text{mono}}$ computations. Similarly, updating **TreeNode** takes $r \cdot C_{\text{seed}}$ computations. Therefore, the total cost is $(r + w) \cdot \log_2 N \cdot C_{\text{check}} + r \cdot C_{\text{seed}} + w \cdot C_{\text{mono}}$.

Observe that the cost of the countermeasure may vary with the value of r , which is the number of seeds published from *Seed Tree*. We have benchmarked the performance of the LESS-sign algorithm with and without our countermeasure for all parameter sets of LESS in Table 3. As we can see, including our countermeasure does not degrade the performance of the LESS-sign algorithm.

For benchmarking, we have used an HP Elite Tower 600 G9 Desktop with an Intel Core i7-12700 CPU running at 2.1 GHz and 32 GB physical memory, which was running Ubuntu 22.04.4 LTS. The test codes were executed on a single core with Turbo Boost and hyperthreading disabled.

Table 3. LESS-sign performance comparison with our countermeasure against original LESS implementation.

Security Level	Parameter Set	Average cpucycles ($\times 10^6$ cycles)	
		Our Countermeasure	Original LESS
1	LESS-1b	1162.31	1162.19
	LESS-1i	1148.03	1147.94
	LESS-1s	931.57	931.72
3	LESS-3b	9563.01	9564.23
	LESS-3s	11285.14	11283.65
5	LESS-5b	44031.11	44036.56
	LESS-5s	29544.22	29542.31

7 Discussion and Future Direction

In this study, we have assumed a single-fault model where an attacker can only inject a fault in one single location. The countermeasure we have provided is based on that assumption. We emphasize the necessity of future investigations into higher-order fault models, side-channel attacks using power, electromagnetic radiation [20, 28], and combined (side-channel assisted fault attack) attack. This study is the first research study enhancing the security of the digital signature scheme LESS and CROSS against a broader spectrum of fault attacks. LESS has $(s - 1)$ secret monomial matrices, and we’ve shown that one pair can recover some information about one secret matrix. So, we need multiple targeted pairs to retrieve all secret matrices. This number of required pairs depends on various parameters. Therefore, we require more than one effective faulted signature for some parameter sets of LESS. For CROSS, there’s only one secret \mathbf{e} for all the parameter sets. It can be recovered with just one targeted pair.

In this work, we have done a fault analysis of the LESS [35] signature scheme that has been submitted to NIST. However, the authors of LESS have updated the scheme in the LESS project’s site [21]. We observe that our mentioned attack surface, *i.e.*, the computation of *TreeNode* by using the function *SeedTreePaths*, are present there too. So, our attack is still applicable to their updated version. Another code-based signature scheme MEDS (Matrix Equivalence Digital Signature) [12] based on the zero-knowledge protocol. Like LESS and CROSS, the Sign algorithm of MEDS uses a similar tree construction to reduce the signature size. In this case, the response $(\tilde{\mathbf{A}}_i \text{ or } \mathbf{Q} \cdot \tilde{\mathbf{A}}_i)$ is constructed depending on some fixed weight digest vector \mathbf{d} , where \mathbf{Q} is a secret component. It involves the same seed tree and *Reference Tree* to store some seeds corresponding to the response $\tilde{\mathbf{A}}_i$ in a similar manner. So, the same attack model can also be applied to the MEDS signature scheme. However, we have not completely

analyzed how many faulted signatures are needed to find the entire secret. We left this part for future work.

We have shown a fault detection method where we have fixed a position $\mathbf{x}[i]$ of the **Reference Tree** and injected fault at that location. The detection method in Sect. 3.4 can detect a successful and effective fault at the location $\mathbf{x}[i]$ for any chosen i , where $1 \leq i < 4l - 1$. Moreover, this method can determine the occurrence of an effective fault at any arbitrary location within the reference tree by applying the detection procedure for each $1 \leq i < 4l - 1$. Given that $l \in \{128, 512, 1024\}$ (according to Table 1), this approach is computationally feasible. However, a mathematical analysis for this scenario has not been included and is left for future work.

Acknowledgements. This work was partially supported by Horizon 2020 ERC Advanced Grant (101020005 Belfort), CyberSecurity Research Flanders with reference number VR20192203, BE QCI: Belgian-QCI (3E230370) (see beqci.eu), Intel Corporation, Secure Implementation of Post-Quantum Cryptosystems (SECPQC) DST-India and BELSPO. Angshuman Karmakar is funded by FWO (Research Foundation - Flanders) as a junior post-doctoral fellow (contract number 203056/1241722N LV). Pujá Mondal is supported by C3iHub, IIT Kanpur. Supriya Adhikary is supported by the Prime Minister’s Research Fellowship (PMRF), India.

Supplementary Material

A Comparison of LESS with Other Code-Based Signature Schemes

Table 4 compares the key sizes and performance of LESS with other code-based digital signature schemes submitted to NIST’s additional call for digital signatures [25].

B Verification Algorithm of LESS

The verification algorithm in Algorithm 10 takes a message m and signature $\tau = (\textit{salt}, \textit{cmt}, \textit{TreeNode}, \textit{rsp})$ and the public key PK as inputs and returns 1, if the τ is a valid signature of the message m otherwise, it will return 0.

Table 4. Comparison of code-based signature schemes in terms of performance and size.

Category	Scheme	Performance (M Cycle)		Size (Bytes)	
		Sign.	Verify	Signature	Public Key
Level I	WAVE [36]	1161	205.9	822	3677390
	MEDS [12]	518.1	515.6	9896	9923
	CROSS [34]	22	10.3	10304	61
	LESS [35]	263.6	271.4	5325	98202
Level III	WAVE [36]	3507	464.1	1249	7867598
	MEDS [12]	1467	1462	41080	41711
	CROSS [34]	46.5	18.3	23407	91
	LESS [35]	2446.9	2521.4	14438	70554
Level V	WAVE [36]	7397	813.3	1644	13632308
	MEDS [12]	1629.8	1612.6	132528	134180
	CROSS [34]	74.8	26.1	43373	121
	LESS [35]	10212.6	10458.8	26726	132096

Algorithm 10. LESS_Vrfy(m, τ, PK)

Input: A message m , the public key PK and the signature $\tau = (\text{salt}, \text{cmt}, \text{TreeNode}, \text{rsp})$.

Output: It will return 1, if (m, τ) is a valid message signature pair; Otherwise, return 0.

```

1:  $d' \leftarrow \text{CSPRNG}(\text{cmt}, \mathbb{S}_{t,w})$ 
2: for  $i = 0; i < t; i = i + 1$  do
3:    $d'[i] = 0? f'[i] = 0 : f'[i] = 1$ 
4:  $ESEED' \leftarrow \text{regenerate\_leaves}(\text{salt}, \text{TreeNode}, f')$ 
5:  $G_0 \leftarrow \text{CSPRNG}(gseed, \mathbb{S}_{\text{RREF}}), k = 0$ 
6: for  $i = 1; i < t; i = i + 1$  do
7:   if  $d'[i] = 0$  then
8:      $\tilde{Q}'_i \leftarrow \text{CSPRNG}(ESEED'[i], M_n)$ 
9:      $(\tilde{Q}'_i, \tilde{V}'_i) \leftarrow \text{PreparedDigestInput}(G_0, \tilde{Q}'_i)$ 
10:  else
11:     $j = d'[i]$ 
12:     $G_j \leftarrow \text{ExpandRREF}(\text{PK}[j])$ 
13:     $Q^* \leftarrow \text{ExpandToMonomAction}(\text{rsp}[k])$ 
14:    Compute  $J \leftarrow \{\alpha_i : Q^*[\alpha_i, *] = 0\}$ 
15:     $\hat{G} \leftarrow (G_j Q^* | G_j[*], J)$ 
16:     $(\hat{G}, \text{pivot\_column}) \leftarrow \text{RREF}(\hat{G})$ 
17:     $NP = 0, \bar{V}'_i = \mathbf{0}$ 
18:    for  $c = 0; c < n; c = c + 1$  do
19:      if  $\text{pivot\_column}[c] = 0$  then
20:         $\bar{V}'_i \leftarrow \text{LexMin}(\hat{G}, \bar{V}'_i, NP, c), NP = NP + 1$ 
21:         $\bar{V}'_i \leftarrow \text{LexSortCol}(\bar{V}'_i), k = k + 1$ 
22:  $\text{cmt}' \leftarrow \text{H}(\bar{V}'_0, \dots, \bar{V}'_{t-1}, m, \text{len}, \text{salt})$ 
23:  $\text{cmt} = \text{cmt}'? \text{Return } 1 : \text{Return } 0$ 

```

References

1. Abdalla, M., An, J.H., Bellare, M., Namprempre, C.: From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security. In: Knudsen, L.R. (ed.) *Advances in Cryptology - EUROCRYPT 2002*, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings. *Lecture Notes in Computer Science*, vol. 2332, pp. 418–433. Springer (2002). https://doi.org/10.1007/3-540-46035-7_28,
2. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D.: Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. Online. Accessed 26th January, 2024 (2022), <https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf>
3. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehle, D.: *CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation* (version 3.02). Online (2021), <https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf>
4. Beckwith, L., Wallace, R., Mohajerani, K., Gaj, K.: A High-Performance Hardware Implementation of the LESS Digital Signature Scheme. In: Johansson, T., Smith-Tone, D. (eds.) *Post-Quantum Cryptography - 14th International Workshop, PQCrypto 2023*, College Park, MD, USA, August 16-18, 2023, Proceedings. *Lecture Notes in Computer Science*, vol. 14154, pp. 57–90. Springer (2023). https://doi.org/10.1007/978-3-031-40003-2_3,
5. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS+ Signature Framework. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. p. 2129-2146. CCS '19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3319535.3363229>,
6. Beullens, W.: Breaking Rainbow Takes a Weekend on a Laptop. *Cryptology ePrint Archive*, Paper 2022/214 (2022), <https://eprint.iacr.org/2022/214>,
7. Biasse, J.F., Micheli, G., Persichetti, E., Santini, P.: LESS is More: Code-Based Signatures Without Syndromes. In: Nitaj, A., Youssef, A. (eds.) *Progress in Cryptology - AFRICACRYPT 2020*, pp. 45–65. Springer International Publishing, Cham (2020)
8. Breier, J., Hou, X.: How Practical are Fault Injection Attacks, Really? *Cryptology ePrint Archive*, Paper 2022/301 (2022), <https://eprint.iacr.org/2022/301>,
9. Bruinderink, L.G., Pessl, P.: Differential Fault Attacks on Deterministic Lattice Signatures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**(3), 21–43 (2018). <https://doi.org/10.13154/TCHES.V2018.I3.21-43>,
10. Castryck, W., Decru, T.: An Efficient Key Recovery Attack on SIDH. In: *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 V. p. 423-447. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30589-4_15,
11. Cho, J., No, J.S., Lee, Y., Koo, Z., Kim, Y.S.: Enhanced pqsigRM: Code-Based Digital Signature Scheme with Short Signature and Fast Verification for Post-Quantum Cryptography. *Cryptology ePrint Archive*, Paper 2022/1493 (2022), <https://eprint.iacr.org/2022/1493>,

12. Chou, T., Niederhagen, R., Persichetti, E., Randrianarisoa, T.H., Reijnders, K., Samardjiska, S., Trimoska, M.: Take your MEDS: Digital Signatures from Matrix Code Equivalence. *Cryptology ePrint Archive, Paper 2022/1559* (2022), <https://eprint.iacr.org/2022/1559>,
13. Clavier, C.: Secret External Encodings Do Not Prevent Transient Fault Analysis. In: Paillier, P., Verbauwhede, I. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings. Lecture Notes in Computer Science*, vol. 4727, pp. 181–194. Springer (2007). https://doi.org/10.1007/978-3-540-74735-2_13,
14. Ding, J., Schmidt, D.: Rainbow, a New Multivariable Polynomial Signature Scheme. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) *Applied Cryptography and Network Security*, pp. 164–175. Springer, Berlin Heidelberg, Berlin, Heidelberg (2005)
15. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehle, D.: CRYSTALS – Dilithium: Digital Signatures from Module Lattices. *Cryptology ePrint Archive, Paper 2017/633* (2017), <https://eprint.iacr.org/2017/633>,
16. Galbraith, S.D., Petit, C., Silva, J.: Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems. In: Takagi, T., Peyrin, T. (eds.) *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. Lecture Notes in Computer Science*, vol. 10624, pp. 3–33. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_1,
17. Genêt, A., Kannwischer, M.J., Pelletier, H., McLaughlan, A.: Practical Fault Injection Attacks on SPHINCS. *IACR Cryptol. ePrint Arch.* p. 674 (2018), <https://eprint.iacr.org/2018/674>
18. Groot Bruinderink, L., Hülsing, A., Lange, T., Yarom, Y.: Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2016*, pp. 323–345. Springer, Berlin Heidelberg, Berlin, Heidelberg (2016)
19. Jao, D., Feo, L.D.: Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In: *PQCrypto. Lecture Notes in Computer Science*, vol. 7071, pp. 19–34. Springer (2011)
20. Kundu, S., Chowdhury, S., Saha, S., Karmakar, A., Mukhopadhyay, D., Verbauwhede, I.: Carry Your Fault: A Fault Propagation Attack on Side-Channel Protected LWE-based KEM. *IACR Cryptol. ePrint Arch.* p. 1674 (2023), <https://eprint.iacr.org/2023/1674>
21. LESSProjectSite: LESS project (2023), <https://www.less-project.com/>
22. Meyer, C.: *Matrix Analysis and Applied Linear Algebra. Other Titles in Applied Mathematics*, Society for Industrial and Applied Mathematics (2000), <https://books.google.co.in/books?id=HoNgdpJWnWMC>
23. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) *Advances in Cryptology – CRYPTO ’85 Proceedings*, pp. 417–426. Springer, Berlin Heidelberg, Berlin, Heidelberg (1986)
24. Mondal, P., Kundu, S., Bhattacharya, S., Karmakar, A., Verbauwhede, I.: A practical key-recovery attack on LWE-based key-encapsulation mechanism schemes using Rowhammer. *CoRR abs/2311.08027* (2023). <https://doi.org/10.48550/ARXIV.2311.08027>,
25. NIST: NIST Announces Additional Digital Signature Candidates for the PQC Standardization Process. Online. Accessed 26th January, 2024 (2023), <https://csrc.nist.gov/news/2023/additional-pqc-digital-signature-candidates>

26. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical CCA2-Secure and Masked Ring-LWE Implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**(1), 142–174 (2018). <https://doi.org/10.13154/TCHES.V2018.I1.142-174>,
27. Persichetti, E., Santini, P.: A New Formulation of the Linear Equivalence Problem and Shorter LESS Signatures, pp. 351–378 (12 2023). https://doi.org/10.1007/978-981-99-8739-9_12
28. Pessl, P., Prokop, L.: Fault Attacks on CCA-secure Lattice KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**(2), 37–60 (2021). <https://doi.org/10.46586/TCHES.V2021.I2.37-60>,
29. Poddebniak, D., Somorovsky, J., Schinzel, S., Lochter, M., Rösler, P.: Attacking Deterministic Signature Schemes using Fault Attacks. *Cryptology ePrint Archive*, Paper 2017/1014 (2017), <https://eprint.iacr.org/2017/1014>,
30. Proos, J., Zalka, C.: Shor’s discrete logarithm quantum algorithm for elliptic curves. *Quantum Inf. Comput.* **3**(4), 317–344 (2003). <https://doi.org/10.26421/QIC3.4-3>,
31. Qiao, R., Seaborn, M.: A new approach for rowhammer attacks. In: 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 161–166 (2016). <https://doi.org/10.1109/HST.2016.7495576>
32. Rennie, B., Dobson, A.: On stirling numbers of the second kind. *Journal of Combinatorial Theory* **7**(2), 116–121 (1969). [https://doi.org/10.1016/S0021-9800\(69\)80045-1](https://doi.org/10.1016/S0021-9800(69)80045-1), <https://www.sciencedirect.com/science/article/pii/S0021980069800451>
33. Rivest, R.L., Shamir, A., Adleman, L.M.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* **21**(2), 120–126 (1978). <https://doi.org/10.1145/359340.359342>
34. Schemes, N.P.Q.C.D.S.: CROSS: Codes and Restricted Objects Signature Scheme - Specification Document (Jan 2022), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/CROSS-spec-web.pdf>
35. Schemes, N.P.Q.C.D.S.: Less: Linear equivalence signature scheme - Specification Document (Jan 2022), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/less-spec-web.pdf>
36. Schemes, N.P.Q.C.D.S.: WAVE: Round 1 Submission - Specification Document (Jan 2022), <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/wave-spec-web.pdf>
37. Shor, P.W.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994. pp. 124–134. IEEE Computer Society (1994). <https://doi.org/10.1109/SFCS.1994.365700>,
38. Sullivan, G.A., Sippe, J., Heninger, N., Wustrow, E.: Open to a fault: On the passive compromise of TLS keys via transient errors. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 233–250. USENIX Association, Boston, MA (Aug 2022), <https://www.usenix.org/conference/usenixsecurity22/presentation/sullivan>
39. Xagawa, K., Ito, A., Ueno, R., Takahashi, J., Homma, N.: Fault-Injection Attacks Against NIST’s Post-Quantum Cryptography Round 3 KEM Candidates. In: Tibouchi, M., Wang, H. (eds.) *Advances in Cryptology - ASIACRYPT 2021*, pp. 33–61. Springer International Publishing, Cham (2021)



More Vulnerabilities of Linear Structure Sbox-Based Ciphers Reveal Their Inability to Resist DFA

Amit Jana^(✉), Anup Kumar Kundu, and Goutam Paul

Cryptology and Security Research Unit, Indian Statistical Institute, Kolkata,
203 Barrackpore Trunk Road, Kolkata 700108, India
janaamit001@gmail.com, goutam.paul@isical.ac.in

Abstract. At Asiacrypt 2021, Baksi et al. introduced DEFAULT, the first block cipher designed to resist differential fault attacks (DFA) at the algorithm level, boasting of 64-bit DFA security. However, during Eurocrypt 2022, Nageler et al. presented a DFA attack that exposed vulnerabilities in the claimed DFA security of DEFAULT, reducing it by up to 20 bits in the case of the simple key schedule and even allowing for unique key recovery in the presence of rotating keys. In this work, we compute deterministic differential trails for up to five rounds, injecting around 5 faults into the simple key schedule for key recovery, recovering equivalent keys with just 36 faults in the DEFAULT-LAYER, and introducing a generic DFA approach suitable for round-independent keys within the DEFAULT cipher. These results represent the most efficient key recovery achieved for the DEFAULT cipher under DFA attacks so far. Additionally, we introduce a novel fault attack called the Statistical-Differential Fault Attack (SDFA), specifically tailored for linear-structured SBox-based ciphers like DEFAULT. This technique is successfully applied to BAKSHEESH, resulting in a nearly unique key recovery. Our findings emphasize the vulnerabilities present in linear-structured SBox-based ciphers and underscore the challenges in establishing robust DFA protection for such cipher designs.

Keywords: Differential Fault Attack · Statistical Fault Attack · Statistical-Differential Fault Attack · DEFAULT · DFA Security

1 Introduction

The differential fault attack (DFA) is a powerful physical attack that poses a significant threat to symmetric key cryptography. Introduced in the field of block ciphers by Biham and Shamir [10], DFA [23, 25, 33] has proven to be capable of compromising the security of many block ciphers that were previously considered secure against classical attacks. While nonce-based encryption schemes can automatically prevent DFA attacks by incorporating nonces in encryption queries, the threat of DFA [19, 28] still persists in designs with a parallelism

degree greater than 2. Additionally, DFA [17, 18, 27] can pose a significant risk to nonce-based designs in the decryption query. In essence, DFA represents a significant challenge for cryptographic implementations whenever an attacker can induce physical faults. In response to this threat, the research community has focused on proposing countermeasures to enhance the DFA resistance of ciphers.

Countermeasures against fault injection attacks can be classified into two main categories. The first category focuses on preventing faults from occurring by utilizing specialized devices. The second category focuses on mitigating the impact of faults through redundancy or secure protocols. Countermeasures that mitigate the effects of fault injection attacks utilize redundancy for protection. These countermeasures can be classified into three categories based on where the redundancy is introduced: cipher level (no redundant computation), using a separate dedicated device, and incorporating redundancy in computation (commonly achieved through circuit duplication). Additionally, protocol-level techniques can also be employed to enhance fault protection.

Most of the countermeasures against attacks on cryptographic primitives, modes of operation, and protocols are focused on implementation-level defenses without requiring changes to the underlying cryptographic algorithms or protocols themselves. One effective countermeasure against DFA is to introduce redundancy into the system so that it can still function even if some faults or errors are introduced. Another countermeasure is to use error detection and correction codes. These codes can detect when errors or faults have occurred and correct them before they affect the output. Recent cryptographic designs propose primitives with built-in features to enable protected implementations against DFA attacks. For instance, FRIET [31] and CRAFT [9] are efficient and provide error detection. DEFAULT [4] is a more radical approach, aiming to prevent DFA attacks through cipher-level design. A brief survey on fault attacks and their countermeasures in symmetric key cryptography can be found in [3].

DEFAULT is a block cipher design proposed by Baksi *et al.* at Asiacrypt 2021 protects against DFA attacks at the cipher level. The primary component of the DFA protection layer in DEFAULT (called the DEFAULT-LAYER) is a weak class linear structure (LS) based substitution boxes (SBox), which behave like linear functions in some aspects. The idea behind the DEFAULT design is that strong non-linear SBoxes are more resistant against classical differential attacks (DA), but weaker against DFA attacks. Conversely, weaker non-linear SBoxes are more resistant against DFA attacks but weaker against DA. Simply speaking, the DEFAULT cipher is a combination of DEFAULT-LAYER (where rounds use LS SBoxes) and DEFAULT-CORE (where rounds use non-LS SBoxes). To address this trade-off, DEFAULT maintains the main cipher, which is presumed secure against classical attacks, and adds two keyed permutations as additional layers before and after it. These keyed permutations have a unique structure that makes DFA non-trivial on them, resulting in a DFA-resistant construction. The SBox in DEFAULT-LAYER features three non-trivial LS elements, resulting in specific inputs/outputs becoming differentially equivalent, including the associated keys. As a result, attackers cannot learn more than half of the key bits by attacking the

SBox layer. The designers claim that by using DFA, an adversary can only recover 64 bits out of a 128-bit key, leaving a remaining key space of 2^{64} candidates that is difficult to brute-force. For even more security, the design approach can be scaled for a larger master key size. In their initial design [5], the authors first propose the simple key schedule function where the master key is used throughout each round in the cipher. Then in [4] the authors update the simple key schedule by recommending to use of the rotating key schedule function in the cipher to make it a more DFA secure cipher.

In [24], the authors initially demonstrate the vulnerability of the simple key schedule of the DEFAULT cipher to DFA attacks. They highlight that this attack can retrieve more key information than what the cipher’s designers claimed, surpassing the 64-bit security level. The authors also present a method to retrieve the key in the case of a rotating key schedule by exploiting faults to create an equivalent key and then targeting the DEFAULT-CORE to recover the actual key. However, their attack on the simple key schedule does not achieve unique key recovery even with an increased number of injected faults. Moreover, as described in [12], this work presents a differential fault attack on the DEFAULT cipher under the simple key schedule, but it is worth noting that this attack is not applicable to the modified version of the cipher employing a key scheduling algorithm.

In recent times, Baksi et al. introduced a new lightweight block cipher based on linear structure (LS SBox) principles, called BAKSHEESH [6]. Similar to the DEFAULT-LAYER, which incorporates three non-trivial LS elements within its SBox, this newly introduced design features only one non-trivial LS element, resulting in a DFA security level of 2^{32} . Although the designers have not explicitly claimed any DFA security, we find it pertinent to conduct a comprehensive investigation into its DFA security, given its alignment with the LS SBox-based design paradigm. To the best of our knowledge, so far there is only a single analysis using the truncated impossible differential on the cipher [21].

1.1 Our Contributions

In this paper, we make several contributions in the field of fault attacks on LS SBox-based ciphers: DEFAULT and BAKSHEESH. Firstly, we demonstrate the vulnerability of the DEFAULT cipher to DFA attacks under bit-flip fault models, specifically targeting the simple key schedule. Our approach effectively reduces the key space with a minimal number of injected faults, surpassing the performance of previous attacks. To achieve this, we propose novel techniques to uniquely compute the differential trail up to five rounds by analyzing the ciphertext differences. These techniques enable us to filter the intermediate rounds and further reduce the key space.

Furthermore, we extend our analysis to the rotating key schedule and showcase the efficiency of our approach in reducing the key space to a unique solution with a minimal number of faults. Additionally, we present a general framework for computing equivalent keys of the DEFAULT-LAYER cipher. By applying this

framework, we demonstrate the efficacy of DFA attacks on rotating key schedules with significantly fewer faults.

Moreover, we introduce a new attack called the *Statistical-Differential Fault Attack* under the bit-set fault model. This attack efficiently recovers the round keys of the DEFAULT cipher, even when the keys are independently chosen from random sources.

Finally, we applied our proposed DFA attack to another linear-structured SBox-based cipher, BAKSHEESH, efficiently recovering its master key uniquely. Likewise, under the bit-set fault model, the SDFA attack can be effectively applied to nearly retrieve its key uniquely.

To summarize our contributions, we provide a concise performance comparison between our enhanced attacks and previous attack methods in Table 1. Our work represents a substantial advancement in the field of fault attacks on LS SBox-based ciphers, notably the DEFAULT and BAKSHEESH ciphers, by introducing a highly effective key recovery strategy.

Table 1. Differential Fault Attacks on DEFAULT with Different Key Schedules

Cipher	Key Schedule	Relevant Works	Attack Strategy	Results		References
				# of Faults	Keyspace	
DEFAULT	Simple	Nageler <i>et al.</i>	Enc-Dec IC-DFA	16	2^{39}	[24, Section 6.1]
			Multi-round IC-DFA	16	2^{20}	[24, Section 6.2]
		This Work	Second-to-Last Round Attack	64	2^{32}	Section 3.1.2
			Third-to-Last Round Attack	34	1	Section 3.1.3
			Fourth-to-Last Round Attack	16	1	Section 3.1.4
			Fifth-to-Last Round Attack	5	1	Section 3.1.5
	SDFA	[64, 128]	1	Section 4.2		
	Rotating	Nageler <i>et al.</i>	Generic NK-DFA	$1728 + x$	1	[24, Section 4.3]
			Enc-Dec IC-NK-DFA	$288 + x$	2^{32}	[24, Section 5.1]
			Multi-round IC-NK-DFA	$(84 \pm 15) + x$	1	[24, Section 5.2, 6.3]
		This Work	Third-to-Last Round Attack	$96 + x$	1	Section 3.2.2
			Fourth-to-Last Round Attack	$48 + x$	1	Section 3.2.2
			Fifth-to-Last Round Attack	$36 + x$	1	Section 3.2.2
			SDFA	[64, 128]	1	Section 4.3
BAKSHEESH		Rotating	This Work	Second-to-Last Round Attack	40	1
	Third-to-Last Round Attack			12	1	Section 5.1.3
	SDFA			128	1	Section 5.2

* x represents the number of faults to retrieve the key at the DEFAULT-CORE. We verified that 32 bit-faults at the second-to-last round in DEFAULT-CORE achieve unique key recovery.

2 Preliminaries

In this section, we will introduce the notations that will be utilized throughout the paper. Following that, we will provide descriptions of the DEFAULT and BAKSHEESH ciphers. Subsequently, we will provide a concise overview of DFA attacks, followed by an in-depth discussion of the linear structure (LS) SBox, a

crucial element in designing a block cipher with DFA protection. The following notations are used throughout the paper.

1. $a \oplus b$ denotes the bit-wise \oplus of a and b .
2. \cup, \cap denotes the set union and intersection respectively.
3. R^i denotes the i^{th} round of the corresponding cipher.
4. K^i denotes the i^{th} subkey of the corresponding cipher.
5. $+$ denotes the integer addition.
6. ΔC denotes the ciphertext difference.
7. N_j^i denotes the j^{th} nibble at R^i .
8. K_j^i denotes the j^{th} key-nibble of K^i .

2.1 Description of DEFAULT Cipher

The DEFAULT cipher [4] is a lightweight block cipher with a 128-bit state and key size. It is designed to resist DFA attacks by limiting the amount of key information that can be learned by an attacker. The cipher incorporates two keyed permutations, known as DEFAULT-LAYER, as additional layers before and after the main cipher. The DEFAULT cipher consists of two main building blocks: DEFAULT-LAYER and DEFAULT-CORE. The DEFAULT-LAYER layer protects the cipher from DFA attacks, while the DEFAULT-CORE layer protects against classical attacks. In short, the encryption function of the DEFAULT cipher can be expressed as $Enc = Enc_{DEFAULT-LAYER} \circ Enc_{CORE} \circ Enc_{DEFAULT-LAYER}$.

The DEFAULT cipher employs a total of 80 rounds, with the DEFAULT-LAYER function being applied 28 times and the DEFAULT-CORE function being applied 24 times. Each round function consists of a structured 4-bit SBox layer, a permutation layer, an add round constant layer, and an add round key layer. The DEFAULT-LAYER function utilizes a linear structured SBox, while the DEFAULT-CORE function utilizes a non-linear structured 4-bit SBox. In the following subsections, we will discuss each component of the DEFAULT cipher in detail.

SBoxes. The DEFAULT-LAYER layer of the DEFAULT cipher utilizes a 4-bit Linear Structured SBox, denoted as S . The SBox consists of four linear structures: $0 \rightarrow 0, 6 \rightarrow a, 9 \rightarrow f$, and $f \rightarrow 5$. The definition of a linear structure can be found in Definition 1. Similarly, the DEFAULT-CORE layer uses another SBox, denoted as S_c . Both the tables and their corresponding DDT’s are given below.

$x : 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ a\ b\ c\ d\ e\ f$	$x : 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ a\ b\ c\ d\ e\ f$
$S(x) : 0\ 3\ 7\ e\ d\ 4\ a\ 9\ c\ f\ 1\ 8\ b\ 2\ 6\ 5$	$S_c(x) : 1\ 9\ 6\ f\ 7\ c\ 8\ 2\ a\ e\ d\ 0\ 4\ 3\ b\ 5$

Definition 1 (Linear Structure). For $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$, an element $a \in \mathbb{F}_2^n$ is called a linear structure of F if for some constant $c \in \mathbb{F}_2^n$, $F(x) \oplus F(x \oplus a) = c$ holds $\forall x \in \mathbb{F}_2^n$.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16																0	16															
1		8							8								1			2		2	2	2	2	2		2	2				
2			8							8				8			2				4	4							4	4			
3				8											8		3		2	2	2	2		2	2		2	2					
4					8									8			4				4	4							4	4			
5						8									8		5			2		2	2	2	2	2		2	2				
6											16						6		4	4				4	4								
7			8						8								7		2	2	2	2		2	2		2	2					
8							8					8					8			4		4			4		4	4					
9															16		9			2	2	2	2		2	2		2	2				
a		8										8					a		4					4	8								
b			8						8								b		2	2	2	2	2	2		2							
c				8									8				c		4	4				4	4								
d					8					8							d		2	2	2	2		2	2		2	2					
e						8							8				e		4					4	8								
f							16										f		2	2	2	2	2	2		2							

Permutation Bits. The DEFAULT cipher incorporates the GIFT-128 permutation (P) in each of its rounds, which is derived from the GIFT [7] cipher. In the permutation layer of the GIFT cipher, there are two versions: one with 4 Quotient-Remainder groups for the 64-bit version, and another with 8 Quotient-Remainder groups for the 128-bit version. It is worth noting that these 8 Quotient-Remainder groups do not diffuse over themselves for 2 rounds.

Add Round Constants. For DEFAULT cipher, a round constant of 6-bits are XORed with the indices 23, 19, 15, 11, 7 and 3 respectively at each of the rounds. Along with this, the bit index 127 is flipped at each round to modify the state bits.

Add Round Key. The round key for DEFAULT cipher is 128 bits in length. In the first preprint version of DEFAULT, a simple key schedule was used where all the round keys were the same as the master key for each round. However, in a later version, a stronger key schedule was proposed to enhance security against DFA attacks. In this updated version, the authors introduced an idealized key schedule where each round key is independent of the others. Although this idealized scheme requires 28×128 key bits to encrypt 128 bits of state using the DEFAULT cipher, it is not practical. To address this, the authors employed an unkeyed function R to generate four different round keys K_0, \dots, K_3 , where $K_0 = K$ and $K_i = R^4(K_{i-1})$ for $i \in 1, 2, 3$. These four round keys are then used periodically for each round to encrypt the plaintext.

2.2 Specification of BAKSHEESH

BAKSHEESH [6] is a lightweight block cipher designed to process 128-bit plaintexts. It is based on the GIFT-128 [7] cipher, featuring 35 rounds of encryption.

Within its design, BAKSHEESH employs a 4-bit substitution-permutation box (SBox) with a non-linear LS element. The round function of BAKSHEESH comprises four operations: SubCells—applying a 4-bit linear structured SBox to the state, PermBits—permuting the bits of the state (similar to GIFT-128), AddRoundConstants—ing a 6-bit constant and an additional bit to the state (similar to GIFT-128), and AddRoundKey—ing the round key with the state. The SBox and its DDT are sketched below.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16															
1	4		4		4				4							
2	4		4		4				4							
3	4 4				4 4				4 4							
4	4		4		4				4							
5	4 4				4				4							
6	4				4				4 4							
7	4		4		4				4							
8	16															
9	4		4		4				4							
a	4		4		4				4							
b	4		4		4 4				4 4							
c	4		4				4		4							
d	4				4		4 4									
e	4		4		4				4							
f	4		4		4				4							

BAKSHEESH exhibits a single linear structure at 8. Additionally, concerning the round keys, the first round key matches the master key, and subsequent round keys are generated with a 1-bit right rotation. More details about the specification of BAKSHEESH cipher can be found in [6].

2.3 Differential Fault Attack

Differential Fault Attack (DFA) is a type of Differential Cryptanalysis that operates in the grey-box model. In this attack, the attacker deliberately introduces faults during the final stages of the cipher to extract the secret component effectively. In contrast, the security of a cipher against Differential Cryptanalysis in the black-box model depends on the probability of differential trails (fixed input/output difference) being as low as possible. However, in DFA, the attacker can introduce differences at the intermediate stages by inducing faults, increasing the trail probability for those rounds significantly. As a result, the attacker can extract the secret component more efficiently than in Differential Cryptanalysis in the black-box model. Finally, estimating the minimum number of faults is crucial in DFA to ensure the attack is both efficient and effective, keeping the search complexity within acceptable limits. To protect ciphers from DFA attacks,

various state-of-the-art countermeasures have been proposed, including the use of dedicated devices or shields that prevent any potential sources of faults. Other countermeasures include the implicit/explicit detection of duplicated computations and mathematical solutions designed to render DFA ineffective or inefficient.

2.4 Revisiting Learned Information via the Linear Structure SBox

A linear structure SBox is a class of permutations that exhibit some properties of linear functions, making them weaker than non-linear permutations in certain aspects. The SBox S used in DEFAULT-LAYER has four linear structures as $\mathcal{L}(S) = \{0, 6, 9, f\}$. According to the DDT of S , the non-trivial linear structures are 6, 9 and f . Similarly, for the inverse SBox S^{-1} , the set of all linear structures of S^{-1} will be $\mathcal{L}(S^{-1}) = \{0, 5, a, f\}$. In their work [4], the designers demonstrate that inducing bit flips before the SBox can yield limited information to attackers, reducing key bits from 4 to 2 during encryption faults. However, in [24], Nageler et al. showed an improved DFA targeting the decryption algorithm, further reducing key bits to 1. This reduction to 2^{32} contradicts the initial claim of 2^{64} key space reduction. Learning key information from a linear structure SBox is non-trivial, and previous works lack detail on this aspect. This section revisits how attackers can glean key information from faults injected during both encryption and decryption queries at the SBox.

Learned Information from S/S^{-1} . Suppose that (x_0, x_1, x_2, x_3) and (y_0, y_1, y_2, y_3) are respectively the bit-level input and output of SBox S . Similarly, (y_0, y_1, y_2, y_3) and (x_0, x_1, x_2, x_3) are the input and output of S^{-1} . Note that, the output of S is same as the input to S^{-1} and vice-versa. Consider a set \mathcal{A} of inputs which satisfy the differential $\alpha \rightarrow \beta$ for the SBox S , i.e., $\mathcal{A} = \{x : S(x) \oplus S(x \oplus \alpha) = \beta\}$. Then, for any $y \in \mathcal{L}(S)$, we have,

$$S(x \oplus y) \oplus S(x \oplus y \oplus \alpha) = (S(x) \oplus S(x \oplus y)) \oplus (S(x \oplus \alpha) \oplus S(x \oplus y \oplus \alpha)) \oplus (S(x) \oplus S(x \oplus \alpha)) = \beta. \quad [\text{As, } (S(x) \oplus S(x \oplus y)) = (S(x \oplus \alpha) \oplus S(x \oplus \alpha \oplus y)).]$$

This result shows that $x \in \mathcal{A} \implies x \oplus y \in \mathcal{A}, y \in \mathcal{L}(S)$. Thus, for any input $x \in \{0, 1, \dots, f\}$, the attacker cannot uniquely identify which among $\{x, x \oplus 6, x \oplus 9, x \oplus f\}$ is the actual input to the SBox. Further, this can be partitioned into four subsets as $\{\{0, 6, 9, f\}, \{1, 7, 8, e\}, \{2, 4, b, d\}, \{3, 5, a, c\}\} = \{\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3\}$. Similarly, for S^{-1} , the partition will be $\{\{0, 5, a, f\}, \{1, 4, b, e\}, \{2, 7, 8, d\}, \{3, 6, 9, c\}\} = \{\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$. The input bit relations of $\mathcal{B}_i/\mathcal{D}_i$'s of S/S^{-1} are denoted by $\mathcal{B}_i^{eq}/\mathcal{D}_i^{eq}$ and given in the table below.

\mathcal{B}_0^{eq}	\mathcal{B}_1^{eq}	\mathcal{B}_2^{eq}	\mathcal{B}_3^{eq}	\mathcal{D}_0^{eq}	\mathcal{D}_1^{eq}	\mathcal{D}_2^{eq}	\mathcal{D}_3^{eq}
$\sum_{i=0}^3 x_i = 0$	$\sum_{i=0}^3 x_i = 0$	$\sum_{i=0}^3 x_i = 1$	$\sum_{i=0}^3 x_i = 1$	$\sum_{i=0}^3 y_i = 0$	$\sum_{i=0}^3 y_i = 1$	$\sum_{i=0}^3 y_i = 1$	$\sum_{i=0}^3 y_i = 0$
$x_0 \oplus x_3 = 0$	$x_0 \oplus x_3 = 1$	$x_0 \oplus x_3 = 0$	$x_0 \oplus x_3 = 1$	$y_0 \oplus y_2 = 0$	$y_0 \oplus y_2 = 1$	$y_0 \oplus y_2 = 0$	$y_0 \oplus y_2 = 1$
$x_1 \oplus x_2 = 0$	$x_1 \oplus x_2 = 1$	$x_1 \oplus x_2 = 1$	$x_1 \oplus x_2 = 0$	$y_1 \oplus y_3 = 0$	$y_1 \oplus y_3 = 0$	$y_1 \oplus y_3 = 1$	$y_1 \oplus y_3 = 1$

For example, consider the SBox S^{-1} (for encryption) with a differential $7 \rightarrow 2$. Then, the number of inputs that satisfy $7 \rightarrow 2$ will be $\mathcal{D}_2 \cup \mathcal{D}_0 =$

$\{0, 5, a, f, 2, 7, 8, d\}$ and hence, the attacker can learn the bit relation of this input set $\mathcal{D}_2 \cup \mathcal{D}_0$ as $\mathcal{D}_2^{eq} \cap \mathcal{D}_0^{eq} \implies y_0 \oplus y_2 = 0$. Similarly, if the differential $7 \rightarrow 4$ happens, then the attacker can learn the bit relation as $\mathcal{D}_1^{eq} \cap \mathcal{D}_3^{eq} \implies y_0 \oplus y_2 = 1$. In this way, for any differential $\alpha \rightarrow \beta$ of S^{-1} , the attacker can learn the bit relation of the inputs that satisfy $\alpha \rightarrow \beta$. Conversely, if we consider the SBox S (for decryption) with differential $\gamma \rightarrow \delta$, the attacker can learn the bit relation from the sets $\mathcal{B}_i, i \in \{0, 1, 2, 3\}$. For example, the inputs to satisfy the differential $2 \rightarrow 7$ will be $\mathcal{B}_2 \cup \mathcal{B}_0$ and thus, input bit relation will be $\mathcal{B}_2^{eq} \cap \mathcal{B}_0^{eq} \implies x_0 \oplus x_3 = 0$. Similarly, for $2 \rightarrow d$, the learned information will be $\mathcal{B}_1^{eq} \cap \mathcal{B}_3^{eq} \implies x_0 \oplus x_3 = 1$.

Consider an encryption query where a nibble difference is injected at the last round before the SBox operation. Let (k_0, k_1, k_2, k_3) be the key \oplus d with the output of SBox and outputs the ciphertext (ignore the linear layer). Now, for each SBox, we are going to combine these learned information for the input/output of S/S^{-1} with the key to learn the corresponding key relation. For example, consider the learned information $y_0 \oplus y_2 = 0$ for a given differential $2 \rightarrow 7$ of S ($7 \rightarrow 2$ for S^{-1}). If c be the non-faulty ciphertext, then we have,

$$c_0 \oplus c_2 = (y_0 \oplus y_2) \oplus (k_0 \oplus k_2) \implies (k_0 \oplus k_2) = (c_0 \oplus c_2) \oplus (y_0 \oplus y_2) = c_0 \oplus c_2.$$

This relation shows that the attacker can learn the key information from the ciphertext relation. In the way, for both encryption and decryption, an attacker can learn key informations for each non-zero differential of S/S^{-1} . In the table below, we summarize the key bits information for both enc/dec which can be learned based on the input difference of S/S^{-1} .

Direction	Learned expression															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Enc (S^{-1})	1	$\sum_{i=0}^3 k_i$	$k_0 \oplus k_2$	$k_1 \oplus k_3$	$k_0 \oplus k_2$	$k_1 \oplus k_3$	1	$\sum_{i=0}^3 k_i$	$\sum_{i=0}^3 k_i$	1	$k_1 \oplus k_3$	$k_0 \oplus k_2$	$k_1 \oplus k_3$	$k_0 \oplus k_2$	$\sum_{i=0}^3 k_i$	1
Dec (S)	1	$\sum_{i=0}^3 k_i$	$k_0 \oplus k_3$	$k_1 \oplus k_2$	$\sum_{i=0}^3 k_i$	$k_1 \oplus k_2$	1	$k_0 \oplus k_3$	$k_0 \oplus k_3$	1	$k_1 \oplus k_2$	$\sum_{i=0}^3 k_i$	$k_1 \oplus k_2$	$k_0 \oplus k_3$	$\sum_{i=0}^3 k_i$	1

3 Our Improvements of DFA on DEFAULT Cipher

In this work, we focus on improving the previously proposed differential fault analysis (DFA) attack on the DEFAULT cipher, specifically on both its simple and rotating key schedules. To enhance this attack, we first introduce a strategy that allows to uniquely determine the internal differential propagation through rounds when faults are injected up to the fifth-to-last rounds. We demonstrate the effectiveness of this method by applying it to the simple key schedule of the DEFAULT cipher and showing that an attacker can recover the key if faults are introduced during the third, fourth, or fifth-to-last rounds. Additionally, we improve the DFA attack on the rotating key schedule of the DEFAULT cipher. Throughout the paper, we use the encryption oracle to inject faults. Overall, our work is focused on fully breaking the DFA security of the DEFAULT cipher under difference-based fault attacks with fewer faults and providing insights into the challenges of using linear structure (LS) substitution boxes (SBox) in block ciphers to achieve cipher-level protection.

Fault Model. In this attack, we consider a fault model where the goal is to induce a precise single bit-flip faults in the cryptographic state nibble during the encryption query. For instance, an attacker might deliberately induce a single bit-flip fault to alter a single bit in the nibble, located just before the input to the SBox operation, at the i^{th} last round of the state during encryption. Achieving this level of precision is feasible in practice, as attackers can employ techniques such as Laser fault injection [2, 15, 30]. These methods offer high accuracy in both space and time. Additionally, electromagnetic (EM) fault injection serves as an alternative method that does not require the de-packaging of the chip. Practical implementation of precise bit-level fault injections has been demonstrated through EM fault injection setups, as illustrated in [8, 20]. Note that this fault model is the same as the one used in the previous work [24].

3.1 Attacks on Simple Key Schedule

In their previous work, Nageler et al. [24] expanded their DFA attack by inducing bit-flip faults across multiple rounds to further reduce the key space. Their strategy involved injecting differences at certain rounds and exploring all possible differential paths through subsequent rounds based on the DDT. By analyzing the distribution of input/output differences at each SBox in subsequent rounds, they conducted differential analysis to recover key bits. However, this approach could not reduce the key space beyond 2^{16} , despite the potential for inducing additional faults¹. In this work, we delved deeper into this issue and devised a novel approach to achieve complete key recovery with significantly fewer faults. Moreover, our proposed attack enables key retrieval with significantly reduced offline computation time compared to previous approaches.

3.1.1 Faults at the Last Round

In this attack scenario, the attacker needs to inject faults and analyze each of the 32 Substitution Boxes (SBox) independently. As per the designers' claim, injecting faults at each SBox nibble can reduce the search space from 2^4 to 2^2 at most, resulting in a total search complexity of $4^{32} = 2^{64}$. However, in [24], the authors further reduce the SBox nibble space to 2 by injecting faults at the decryption algorithm. Specifically, the authors demonstrate how to derive three linearly independent equations for each nibble of the key by inducing two and one faults in the encryption and decryption algorithms, respectively. It is worth noting that computing the table [24, Table 3], which calculates the learned information involving the key nibbles for encryption and decryption algorithms,

¹ In [24], for the basic attack implementation in Section 3.3, the authors inject faults at the fourth-to-last round in `attack_r3()` phase (see the implementation). Whereas, in their implementation, for the improved attack in Section 6.2, the authors induce faults at the fifth-to-last round to significantly reduce the number of faults. As, `DEFAULT-LAYER` does not have pre-key whitening, the last round is deterministic for the key analysis using DFA in the decryption query.

is not a straightforward process according to their work. That's why, we provided a more detailed explanation for computing this table in Sect. 2.4.

One approach is, to reduce the keyspace to 2^{64} by inducing $2 \times 32 = 64$ number of bit-flip faults at the last round. Thus, based on the information learned from the table in the previous section, an attacker can learn two bits of information for each nibble in the last round of the DEFAULT-LAYER. However, a more efficient strategy is needed to induce faults further from the last rounds and deterministically obtain information about the input differences of each SBox in the last round. This requires developing a strategy that can deterministically compute the differential path from which the faults are injected. In the upcoming subsections, we will demonstrate that it is possible to uniquely compute the differential path of the DEFAULT-LAYER up to five rounds. By inducing around five bit-flip faults at the fifth-to-last round, we estimate that the keyspace can be reduced to 2^{64} with greater efficiency than the naive approach.

3.1.2 Faults at the Second-to-Last Round

In this attack scenario, we assume that a bit-flip fault is injected at each nibble during the second-to-last round of the DEFAULT-LAYER. As a result, the fault propagation can affect at most four nibbles in the final round of the DEFAULT-LAYER. The DEFAULT-LAYER uses the GIFT-128 bit permutation internally, which has a useful property known as the Quotient-Remainder group structure. At round r , the 32 nibbles of a DEFAULT state are denoted as $S_i^r, i = 0, \dots, 31$ and can be grouped into eight groups $\mathcal{G}_{r_i} = (S_{4i}^r, S_{4i+1}^r, S_{4i+2}^r, S_{4i+3}^r)$ for $i = 0, \dots, 7$. This property states that any group at round r is permuted to a group of four nibbles at round $r + 1$ through a 16-bit permutation, i.e.,

$$\mathcal{G}_{r_i} \xrightarrow{16 \text{ bit permutation}} (S_i^r, S_{i+8}^r, S_{i+16}^r, S_{i+24}^r), i = 0, \dots, 7.$$

The structure of the cipher indicates that a nibble difference at the input of group \mathcal{G}_{r_i} in the second-to-last round spreads differences to at most four nibbles: S_i^{r+1} , S_{i+8}^{r+1} , S_{i+16}^{r+1} , and S_{i+24}^{r+1} in the last round. This observation enables an attacker to uniquely determine the differential path by injecting bit-flip faults in the second-to-last round. The reason for this unique trail computation is that the output difference at the last layer SBox, except at the linear structure points, has only two possible input differences. An incorrect input difference guess will lead to a contradiction by activating the SBox in the wrong group in the second-to-last round. Moreover, this observation allows for the deterministic computation of the differential paths up to five rounds, which we will discuss in the next subsections.

Attack Strategy. To attack the cipher in this scenario, a simple approach is to inject two bit-faults at each nibble in the last round, reducing the keyspace of each nibble to 2^2 , i.e., the overall keyspace is thus reduced to 2^{64} . Then, inject one fault at each nibble in the second-to-last round, reducing the keyspace to 2^{32} . To accomplish this, we first group the 32 nibbles of the state into eight

groups \mathcal{G}_{r_i} , each consisting of four nibbles, and consider the combined keyspace of nibble positions $i, i + 8, i + 16$, and $i + 24$ for each group \mathcal{G}_{r_i} .

For each key in the combined keyspace of \mathcal{G}_{r_i} , we invert two rounds by considering the equivalent key classes of individual nibble positions at the second-to-last round and checking whether they satisfy \mathcal{G}_{r_i} 's input difference at the second-to-last round. By doing this, we can determine the internal state difference between the faulty and non-faulty ciphertexts. It is noteworthy that injecting faults in more than one nibble within \mathcal{G}_{r_i} during encryptions at the second-to-last round can further reduce the keyspace for that group, potentially from 2^{16} to 2^4 . The overall keyspace has now been effectively reduced to $2^{4 \cdot 8} = 2^{32}$, considering 8 groups denoted as \mathcal{G}_{r_i} . Initially, this approach necessitates approximately $2 \times 32 + 32 = 96$ bit-faults to achieve this reduction in the keyspace. However, we have enhanced this attack by introducing faults at the second-to-last round during encryption. Our practical verification shows that injecting faults at each SBox in the second-to-last round, specifically inducing faults at the least significant bits of the nibble (i.e., either at index 1 or index 2), notably reduces the keyspace to 2^{32} . This is because the output difference spread more differences if the input difference is either 2 or 4 (please refer to the DDT table of S in Sect. 2.1). The specific values representing the reduced keyspace for varying numbers of injected faults can be found in Table 2.

3.1.3 Faults at the Third-to-Last Round

In this section, we focus on the keyspace reduction using DFA for three rounds. We introduce a fault at the third-to-last round of the cipher, i.e., at round R^{25} in DEFAULT-LAYER. The attack consists of two phases. In the initial phase, we inject a bit-flip fault at the input to the SBox of the third-to-last round and determine the trail of three rounds uniquely. To achieve this, we compute the input and output differences of every nibble at each round, allowing us to trace the propagation of differences through the backward propagation of the cipher. By carefully analyzing the trail, we can establish a deterministic relationship between the input and the output difference of the SBox, enabling us to deduce the trail with high confidence. In the second phase, we utilize the computed trail to reduce the keyspace of the cipher. With knowledge of the trail, we can target specific nibbles and their corresponding input differences at the last round. By exploiting these input differences, we can perform DFA and significantly reduce the keyspace. This reduction is based on the fact that we now have knowledge of the correlations between the input-output differences and the key bits, allowing us to make informed guesses and narrow down the possible key values.

Deterministic Trail Finding. To compute the faulty differential trail, the attacker first injects faults at the third-to-last round and collects the faulty ciphertexts. Based on the non-faulty and faulty ciphertexts, the attacker derives the ciphertext difference and also has prior knowledge about the input difference to the SBox at the third-to-last round. The steps to compute the unique differential

trail are described in Algorithm 1. In the algorithm, the assumption is that after injecting the faults, the attacker must know both the specific nibble within the state where the fault was injected and the exact faulty value induced at that location, to enable precise fault impact analysis.

For DEFAULT-LAYER, the propagation of faults through rounds is depicted in Fig. 2 using different colors. To better understand the algorithm, consider a fault injected in the 0^{th} Quotient-Remainder group (specifically the 0^{th} nibble in R^{25}), which is colored red in the corresponding figure. For a better understanding of the algorithm, we summarize the steps as follows:

Step 1. In the first step some empty lists are initialized, where \mathcal{A}_{ido}^i stores the input-output difference at round R^i . Whereas \mathcal{D}_{id}^i (resp. \mathcal{D}_{od}^i) stores the input (resp. output) difference from a particular output (resp. input) difference, called the dummy list.

Step 2–7. At R^{25} , the input difference to the SBox at the nibble position $x = 0$ is $(\delta, 0, \dots, 0)$, where δ is known to the attacker. So, we initialize $\mathcal{A}_{ido}^{25}[0][0] = 0$ and $\mathcal{A}_{ido}^{25}[0][x] = 0, x \neq 0$. At this stage, we do not know the output difference, hence for the time being, we assume the output difference corresponding to the nibble position x is $0xf$ (line no. 5). In the next line, after the permutation layer, the state difference after the SBox layer $((0xf, 0, \dots, 0))$ propagates through linear mixing (red lines from R^{25}) so that the active SBox in the next round can be marked (i.e. $N_j^{26}, j \in \{0, 8, 16, 24\}$).

Step 8–9. In the above steps, we cover one round from top to bottom (i.e. from R^{25} to R^{26}). In the steps below, our idea is to go to R^{26} from the ciphertext differences and check whether they map to the sbox positions $N_j^{26}, j \in \{0, 8, 16, 24\}$ or not. For this, we apply the inverse permutation on the ciphertext differences and store them in the list $\mathcal{A}_{iod}^{27}[1]$.

Step 10–19. Here we collect all possible input differences from the DDT table and store it in \mathcal{L}_3 . Now for each difference in \mathcal{L}_3 we check whether, after applying the inverse permutation on the difference, the only active SBox positions are $N_j^{26}, j \in \{0, 8, 16, 24\}$ or not. If it happens, then we assume it as a possible input difference and hence we store it in the corresponding list $\mathcal{A}_{iod}^{27}[0]$ (line 13–18). We store the corresponding inverse permutation of the difference values into the corresponding list (line 19).

Step 20–31. We are in the middle round, i.e., round 26 of the trail. Hence to get the output difference of round 25, we repeat the same procedure as discussed in the steps 2–19. This gives us the unique input-output difference trail for the three rounds of the cipher.

Key Recovery. For each differential trail, we begin by narrowing down the key nibble spaces associated with active SBox in the final round through a comparison of non-faulty and faulty ciphertexts. By introducing two distinct bit differences at each nibble in the final round, we can efficiently reduce the key space K_i^{27} for $i \in \{0, \dots, 31\}$ to 2^2 . Next, we focus on each group \mathcal{G}_i , where

Algorithm 1. DETERMINISTIC COMPUTATION OF THREE ROUNDS DIFFERENTIAL TRAIL

Input: A list of ciphertext difference $\mathcal{L}_{\Delta C}$, faulty value δ , and faulty nibble position x
 Output: Lists of input-output differences $\mathcal{A}_{iod}^{25}, \mathcal{A}_{iod}^{26}$, & \mathcal{A}_{iod}^{27} at the third-to-last, second-to-last, and the last round respectively

- 1: Initialize $\mathcal{L}_1 \leftarrow []$, $\mathcal{A}_{iod}^{25} \leftarrow [[] , []]$, $\mathcal{A}_{iod}^{26} \leftarrow [[] , []]$, $\mathcal{A}_{iod}^{27} \leftarrow [[] , []]$, $\mathcal{D}_{od}^{25} \leftarrow []$, $\mathcal{D}_{id}^{26} \leftarrow []$
- 2: $\mathcal{A}_{iod}^{25}[0] = [0 \text{ for } i \text{ in range}(32)]$
- 3: $\mathcal{A}_{iod}^{25}[0][x] = \delta$
- 4: $\mathcal{D}_{od}^{25} = \mathcal{D}_{id}^{26} = [0 \text{ for } i \text{ in range}(32)] \triangleright$ Dummy output state difference list after the SBox layer
- 5: $\mathcal{D}_{od}^{25}[x] = 0xf$
- 6: $\mathcal{L}_1 = P(\mathcal{D}_{od}^{25})$
- 7: $\mathcal{D}_{id}^{26} = \text{findActiveSBox}(\mathcal{L}_1) \triangleright$ For each non-zero nibble value, this function assign 1 to this nibble index, otherwise it assign to 0
- 8: $\mathcal{A}_{iod}^{27}[1] = P^{-1}(\mathcal{L}_{\Delta C})$
- 9: $\mathcal{L}_3 = [] \triangleright$ Third layer/Last round possible input difference list
- 10: **for** $i = 0$ to $\mathcal{A}_{iod}^{27}[1]$ **do**
- 11: Append $\text{DDT}^{-1}[i]$ to the list \mathcal{L}_3
- 12: $\mathcal{A}_{iod}^{27}[0] = [0 \text{ for } i \text{ in range}(32)]$
- 13: **for** pos, i in $\text{enumerate}(\mathcal{L}_3)$ **do**
- 14: **if** $i \neq [0]$ **then** $dList = [0 \text{ for } i \text{ in range}(32)]$
- 15: **for** dif in i **do**
- 16: $dList[pos] = dif$
- 17: **if** $\text{list_subset}(\text{findActiveSBox}(P^{-1}(dList)), \mathcal{D}_{id}^{26}) == 1$ **then**
- 18: $\mathcal{A}_{iod}^{27}[0][pos] = dif$
- 19: $\mathcal{A}_{iod}^{26}[1] = P^{-1}(\mathcal{A}_{iod}^{27}[0])$
- 20: $\mathcal{L}_2 = [] \triangleright$ Second layer possible input difference list
- 21: **for** i in $P^{-1}(\mathcal{A}_{iod}^{27}[0])$ **do**
- 22: Append $\text{DDT}^{-1}[i]$ to the list \mathcal{L}_2
- 23: $\mathcal{A}_{iod}^{26}[0] = [0 \text{ for } i \text{ in range}(32)]$
- 24: **for** pos, i in $\text{enumerate}(\mathcal{L}_2)$ **do**
- 25: **if** $i \neq [0]$ **then** $dList = [0 \text{ for } i \text{ in range}(32)]$
- 26: **for** dif in i **do**
- 27: $dList[pos] = dif$
- 28: **if** $\text{list_subset}(\text{findActiveSBox}(P^{-1}(dList)), \mathcal{D}_{od}^{25}) == 1$ **then**
- 29: $\mathcal{A}_{iod}^{26}[0][pos] = dif$
- 30: $\mathcal{A}_{iod}^{25}[1] = P^{-1}(\mathcal{A}_{iod}^{26}[0])$
- 31: **return** the lists $\mathcal{A}_{ID}^{27}, \mathcal{A}_{ID}^{26}$ and \mathcal{A}_{ID}^{25}

$i \in \{0, \dots, 7\}$, at the second-to-last round i.e. R^{26} . We combine the keyspaces from the nibble positions $i, i + 8, i + 16$, and $i + 24$ based on the key nibbles of the last round. For each combined key of size 2^8 , we perform the inverse of one round and check the corresponding trail list to determine the resulting differential. At this stage, we use the equivalent key nibble obtained from the reduction at the last round. If the computed differential matches the observed differential, we consider the combined key as a potential key combination. This filtering process is applied to each group at the second-to-last round by guessing 2^8 combined keys and leads to reduce the combined keys to 2^4 ($/2^5$) in the best (/worst) case scenario. Finally, we create combined keyspaces for each even (/odd) position based on the key reductions at the second-to-last round. These correspond to the left (/right) half of the nibbles at the third-to-last round i.e., R^{25} . It is important to note that faults introduced at the sixteen least (/most) significant nibbles of the third-to-last round can affect almost all the even (/odd) position nibbles in the last round. Therefore, in the worst case, filtering the third-to-last round

nibble difference requires trying up to $(2^5)^4 = 2^{20}$ keys. This indicates that the time complexity of this attack is 2^{20} and the memory complexity needs a constant amount of memory. Our practical verification demonstrates that injecting faults at each SBox in the third-to-last round, involving the flipping of the bit at either index 1 or index 2, significantly reduces the keyspace to nearly a unique key. To uniquely recover the key, we verified that 34 bit-flip faults are sufficient. This involves faults at all 32 nibble positions and repeating faults at the 0^{th} and 1^{st} nibble positions with a faulty nibble difference value of 4. The specific values representing the reduced keyspace for varying numbers of injected faults can be found in Table 2. Using 20, 24, 28 and 34 faults the keyspace reduced to 2^{18} , 2^8 , 2^8 and 1 with 40%, 30%, 30% and 100% frequencies respectively. The frequencies are calculated over 1000 experiments for this attack part.

Table 2. Keyspace Reduction with Varying Injected Faults in BAKSHEESH and DEFAULT’s Simple Key Schedule under Differential Fault Attacks

Ciphers	Attack Strategy	Results	
		Number of Faults	Reduced Keyspace
DEFAULT	Faults at the Second-to-Last Round	64	2^{32}
		48	2^{39}
		32	2^{46}
	Faults at the Third-to-Last Round	32	$2^{0.2}$
		28	2^7
		24	2^{14}
	Faults at the Fourth-to-Last Round	16	1
		12	1
		8	2^7
	Faults at the Fifth-to-Last Round	8	1
		6	1
		5	1
BAKSHEESH	Faults at the Second-to-Last Round	48	1
		40	1
		32	2^{32}
	Faults at the Third-to-Last Round	16	1
		12	1
		10	2

3.1.4 Faults at the Fourth-to-Last Round

Here the attacker can inject bit-flip nibble faults at the fourth-to-last round of the cipher, specifically at round R^{24} in DEFAULT-LAYER. The effect of the nibble faults (in the left half i.e., the first 16 least significant nibbles) or right half (i.e., the last 16 most significant nibbles) at the SBox input to the fourth-to-last round can propagate to almost all even or odd nibbles, respectively, at the second-to-last round. Furthermore, at the last round, the differences in even or odd nibbles activate all 32 nibbles in the state.

Deterministic Trail Finding. Assume the initial input difference is given at 0^{th} nibble of R^{24} . Hence after three rounds (which is R^{26}) that nibble differences arise at all even positions in the state before the SBox operations. We have exactly two active even nibbles in each group \mathcal{G}_{r_i} at this round. Consequently, the input nibble difference at each SBox in the last round will no longer be a simple bit difference. Therefore, for each output of SBox at the last round, there are two possible choices of input differences, which may not be in the form of single-bit nibble differences.

To determine the output difference of SBoxes in \mathcal{G}_{r_i} at the second-to-last round, we exhaustively consider all combined input differences corresponding to the positions $i, i+8, i+16,$ and $i+24$ from the last round. We then check whether, after the bit permutation, these differences only go to the even nibble positions in \mathcal{G}_{r_i} , and their corresponding input differences are single-bit differences. This strategy allows us to uniquely identify the output difference of SBoxes in \mathcal{G}_{r_i} at the second-to-last round. The process is described in detail in Algorithm 3 (Appendix A).

Key Recovery. Once unique the trail is computed, we can proceed to reduce the key space by analyzing the last three rounds. First, we iterate exhaustively through the entire key space at the last round for each input-output nibble difference at the fourth-to-last round. Next, we invert the intermediate rounds by using the reduced keys at each round and filter out incorrect keys. By repeating this process for each input-output nibble difference in the last four rounds, we can significantly reduce the key space, approaching a nearly unique solution. Our practical validation confirms that the introduction of 8 bit-faults in each half of the SBox (both left and right) in the fourth-to-last round, achieved by flipping a bit at either index 1 or index 2 (i.e., of faulty value 2 or 4), substantially diminishes the key space, resulting in nearly unique keys. Simply speaking, we need to inject the minimum number of faults in the fourth-to-last round to ensure at least two different input-output differences at each SBox in the last round. This reduces the key space to $(2^{32})^2 = 2^{64}$. Further, the key space is reduced to 2^5 per group in the second-to-last round. Consequently, we try 2^{20} keys for each half (left or right) to filter one SBox in the third-to-last round. Ultimately, based on difference propagation, the key space can be reduced to approximately $2^{20-y} (\approx 2^4)$, where y is the number of different active SBox in the third-to-last round's left/right half. Finally, the filtering in the fourth-to-last round results in unique key retrieval. Therefore, the time and memory complexity of this attack will be 2^{20} and a constant amount of memory, respectively. Detailed information on the reduced key space values corresponding to different fault injection counts is available in Table 2. Using 8, 10, 12 and 16 faults the key space reduced to $2^{6.5}$, 3, 1 and 1 with 40%, 80%, 80% and 100% frequencies respectively. Here also the frequencies are calculated over 1000 experiments.

3.1.5 Faults at the Fifth-to-Last Round

In this section, we discuss how we can deterministically compute the differential trail when injecting faults during the fifth-to-last round (round R^{23}) in the DEFAULT-LAYER cipher. These faults can be injected either in the left half (from nibble positions 0 to 15) or the right half (from positions 16 to 31), affecting either all the even nibble positions or the odd nibble positions in the state at the third-to-last round. An example of fault propagation resulting from a nibble fault in the left half is illustrated in Fig. 3 (Appendix A).

Furthermore, the differences in even/odd nibbles at the third-to-last round activate all the nibbles in the second-to-last round and subsequently in the last round as well. In this attack scenario, we compute the trail for five rounds uniquely and then estimate the number of faults required to recover the key. By doing so, we can significantly reduce the keyspace using a smaller number of faults compared to our previous approaches.

Deterministic Trail Finding. To compute the trail for five rounds when injecting faults at the fifth-to-last round, the approach involves inverting two rounds and then determining the upper three rounds' trails based on the possible differences at the third-to-last round. The objective is to check if these trails satisfy the input difference at the fifth-to-last round. The whole procedure is described in Algorithm 2 (see Appendix A). If the fault is given in the least (resp. most) significant 16 nibbles then after three rounds the even (resp. odd) nibbles becomes active. This is shown in the corresponding figure through two different colors. For a better understanding of the algorithm, we summarize the steps as follows:

Step 1–3. At first, the lists to store the input-output difference for the corresponding round are initialized. Then we have constructed \mathcal{T} to store the possible output difference set after three rounds i.e. R^{25} . $[(T_1)^8, (T_2)^8, (T_1)^8, (T_2)^8]$ (resp. $\mathcal{T}[1]$) denotes the possible input difference after three rounds if the initial input difference is given in the even (resp. odd) nibbles. Here $(T_i)^8$ means the difference in the consecutive 8 nibbles are from the set T_i . This can be viewed in Fig. 3 by red lines for difference in even nibbles and blue for odds.

Step 4–15. Throughout these steps we have taken the all possible input differences corresponding to each i -th Quotient-Remainder group of the last round and store it in the corresponding list. Here we will get 2^8 possible input-output differences for each \mathcal{G}_{r_i} as for each output difference there is only two possible input differences.

Step 16–20. For each of these tuples of the output differences, we invert one more round and collect possible input differences at the end of R^{25} . At step 20, it is checked whether the occurred difference can appear after three rounds if the fault is given in the nibble of R^{23} . This leads to the input output difference list size to approximately 2. In the later step it further filters this to a unique one.

Step 22–23. With this updated list we apply Algorithm 1 and return the trail list which satisfies totally for five rounds.

Key Recovery. In this case we also use the same key recovery process as we have used for four rounds in Sect. 3.1.4. The extra added advantage is that as it’s a five round trail hence we can dive further into one more round and check the input output difference. Therefore, the time and memory complexities of this attack would be same as for the previous fourth-to-last round attack in Sect. 3.1.4.

Our empirical validation strongly supports the notion that introducing a single bit-fault within each of the 8 groups $\mathcal{G}_{r_i}, i \in \{0, 1, \dots, 7\}$ of the SBox, achieved by flipping a bit at either index 1 or index 2, substantially reduces the keyspace, resulting in unique keys. We verified that inducing five faults at the nibble positions 0, 8, 11, 16, and 24 with a faulty value of 4 is sufficient to uniquely recover the key. For comprehensive details regarding the reduced keyspace values associated with varying fault injection counts, we refer to Table 2. Figure 1 shows the distribution of the size of the keyspace after this attack, where the frequency is calculated over 10,000 experiments.

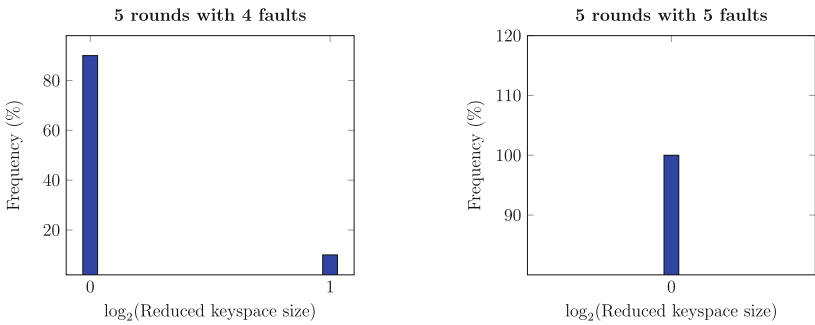


Fig. 1. Distribution of the Reduced Keyspace for the Fifth-to-Last Round Attack

3.2 Attacks on Rotating Key Schedule

In the study presented in [24], the authors introduce the concept of computing an equivalent key, which generates the same ciphertext as the original key for a given plaintext. Building on this notion, the attacker’s strategy involves computing an equivalent key for the DEFAULT-LAYER layer by injecting faults at various rounds. Subsequently, the attacker aims to recover the master key by executing a Differential Fault Analysis (DFA) on the DEFAULT-CORE.

This section begins by explaining how to derive an equivalent key for the DEFAULT-LAYER. We introduce additional methodologies for calculating an equivalent key based on specific properties of the linear structured SBox S . Using this equivalent key, we propose a comprehensive attack strategy based on our deterministic trail computation approach, facilitating the unique recovery of the DEFAULT cipher’s key for different rounds amidst injected faults. This method not only boasts efficient offline computation capabilities but also

requires significantly fewer faults compared to previous attacks. Additionally, we present a versatile attack approach applicable in scenarios where the cipher utilizes multiple round-independent keys.

3.2.1 Exploiting Equivalent Keys

Due to the LS SBox, we know that for any $\alpha \in \mathcal{L}(S) \exists \beta \in \mathcal{L}(S^{-1})$ such that $S(x \oplus \alpha) = S(x) \oplus S(\alpha) = S(x) \oplus \beta, \forall x \in \mathcal{F}_2^4$. Let us define $\mathcal{L}(S, S^{-1}) = \{(\alpha, \beta) : S(x \oplus \alpha) = S(x) \oplus \beta\} = \{(0, 0), (6, a), (9, f), (f, 5)\}$. In another way, we can say that for any $(\alpha, \beta) \in \mathcal{L}(S, S^{-1}), \Pr[\alpha \rightarrow \beta] = 1$. Consider a toy cipher consisting of one DEFAULT-LAYER SBox with a key addition before and after: $y = S(x \oplus k_0) \oplus k_1$, where $k_0, k_1 \in \mathcal{F}_2^4$. Due to the LS SBox, we have for any $(\alpha, \beta) \in \mathcal{L}(S, S^{-1})$,

$$y = S(x \oplus (k_0 \oplus \alpha)) \oplus (k_1 \oplus \beta) = S(x \oplus k_0) \oplus \beta \oplus (k_1 \oplus \beta) = S(x \oplus k_0) \oplus k_1, \forall x \in \mathcal{F}_2^4.$$

This means that if (k_0, k_1) be the actual key used in the toy cipher, then for any $(\alpha, \beta) \in \mathcal{L}(S, S^{-1}), (\hat{k}_0, \hat{k}_1) = (k_0 \oplus \alpha, k_1 \oplus \beta)$ will also be an equivalent key of the toy cipher, i.e., the number of equivalent keys of this toy cipher will be 2^2 . Similarly, any round function of DEFAULT cipher can be think of parallel execution of 32 toy ciphers. Let $k_0 = (k_0^0, k_0^1, \dots, k_0^{31})$ and $k_1 = (k_1^0, k_1^1, \dots, k_1^{31})$ denote the two keys before and after the SBox layer respectively. Then, \forall linear structures $(\alpha^i, \beta^i), i \in \{0, 1, \dots, 31\}$, the number of equivalent keys for the round function of DEFAULT cipher will be $2^{2 \times 32} = 2^{64}$. The various methods for generating equivalent keys of the DEFAULT-LAYER are outlined in [24]. The practical verification to compute the equivalent keys can be found in [1]. Thus, for the DEFAULT-LAYER with four keys (k_0, k_1, k_2, k_3) used in the three round functions, the number of equivalent keys $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$ will be $2^{3 \times 64} = 2^{192}$. As an example, the key $k_0 = 1a5f01b35ef5deea60361f4df591c654$ is equivalent to the key $\hat{k}_0 = 7c3967d53893b88c0650792b93f7a032$ and hence, generate the same ciphertext c corresponds to the message m . Since the keyspace of (k_0, k_1, k_2, k_3) used in the DEFAULT-LAYER is 2^{512} and it has 2^{192} number of equivalent keys for any chosen key, we can further divide the keyspace into $2^{512-192} = 2^{320}$ number of different equivalent key classes.

3.2.2 Generalized Attack Strategy

In this approach, we exploit the fact that injecting two faults at each nibble position in the last round of the encryption process reduces the key nibble space from 2^4 to 2^2 . We iteratively select one key nibble from each reduced set of key nibble values to obtain keys \hat{k}_3, \hat{k}_2 , and \hat{k}_1 . However, at the fourth-to-last round, the key nibbles of k_0 still have 2^2 possible choices. To compute \hat{k}_0 , our strategy involves introducing additional faults at higher rounds and using the other keys \hat{k}_3, \hat{k}_2 , and \hat{k}_1 in conjunction with the deterministic trail computation up to the fifth-to-last round. For instance, if we inject 32 faults at each nibble in the sixth-to-last round of DEFAULT-LAYER, we can trace back from the ciphertext

difference to the fourth-to-last round output difference by applying the equivalent round keys \hat{k}_3 , \hat{k}_2 , and \hat{k}_1 . Based on this fourth-to-last round difference, we can compute the trail for the upper three rounds (from fourth to sixth last rounds) using Algorithm 1.

In the case of the simple key schedule, we have demonstrated that around 32 faults at each nibble in the third-to-last round are adequate for unique key recovery. Similarly, in the scenario described in the previous section, we can uniquely retrieve the key \hat{k}_0 by injecting a suitable number of faults, such as around 12 or 5 faults at the seventh-to-last or eighth-to-last rounds, and deterministically computing the upper trails for four or five rounds using Algorithm 3 or Algorithm 2, respectively. To summarize, the first step requires approximately 256 faults to uniquely select \hat{k}_3 , \hat{k}_2 , and \hat{k}_1 from 2^{64} choices, along with k_0 having 2^{64} possibilities. The recovery of \hat{k}_0 can be accomplished by injecting just 5 extra single bit-flip faults at the eight-to-last round. Consequently, around 261 faults are needed to recover an equivalent key of DEFAULT-LAYER. Once the equivalent key is obtained, the original key can be recovered by injecting faults in the DEFAULT-CORE.

The aim is to explore alternative strategies that can effectively reduce the number of faults required, as opposed to the initial approach of injecting two faults at each nibble in the last four rounds. By leveraging deterministic trail computations, several strategies can be employed to achieve this reduction. These strategies are as follows:

3.2.2.1 Retrieving Equivalent Key Using Three Round Trail Computation. It should be noted that a single bit-flip fault at any nibble can activate at least two nibbles in the next round. By injecting 32 faults at each nibble in the third-to-last round, we can generate at least two differences at each nibble in the second-to-last and last rounds. This allows us to compute \hat{k}_3 and \hat{k}_2 . Then, by injecting another 32 faults at the fifth-to-last round, we can recover \hat{k}_1 and consider the 2^{64} choices of k_0 by computing three-round trails using \hat{k}_3 and \hat{k}_2 . Finally, inducing another 32 faults at the sixth-to-last round, we obtain an equivalent key $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$. In summary, approximately 96 faults are required to recover an equivalent key for DEFAULT-LAYER.

3.2.2.2 Retrieving Equivalent Key Using Four Round Trail Computation. By injecting 32 single bit-flip faults at each nibbles in the fourth-to-last round, we can achieve the generation of at least two different input differences at each nibble in the third-to-last, second-to-last and last rounds which can able to reduce the key nibble space to 2^2 individually. This enables the computation of \hat{k}_3 , \hat{k}_2 and \hat{k}_1 . Additionally, by introducing 8 faults at the seventh-to-last round, we can recover the 2^{64} choices of k_0 by utilizing four-round trails computed using \hat{k}_3 , \hat{k}_2 and \hat{k}_1 . Furthermore, approximately 8 faults at the eighth-to-last round are sufficient to obtain an equivalent key $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$. To summarize, a total of around 48 faults are required to recover an equivalent key for DEFAULT-LAYER.

3.2.2.3 Retrieving Equivalent Key Using Five Round Trail Computation. Like the previous approach, we inject 32 single bit-flip faults at each nibbles in the fifth-to-last round. This ensures the generation of at least two different input differences at each nibble in the fourth-to-last, third-to-last, second-to-last and last rounds respectively and then compute $\hat{k}_3, \hat{k}_2, \hat{k}_1$ and 2^{64} choices of k_0 . Moreover, approximately 4 faults at the tenth-to-last round are sufficient to obtain an equivalent key $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$. As a result, a total of around 36 faults are required to recover an equivalent key for DEFAULT-LAYER. Using 24, 28 and 36 faults we get the equivalent key with 40, 80 and 100% frequencies respectively.

3.2.3 Generic Attack Strategy for More Round Keys

In the scenario where an DEFAULT-LAYER encryption consists of r rounds with $r + 1$ round keys k_0, k_1, \dots, k_r , a simple approach involves injecting two faults at each nibble in the encryption process for each of the r rounds. This allows us to compute r equivalent keys: $\hat{k}_r, \hat{k}_{r-1}, \dots, k_1$. However, the initial key k_0 remains unknown due to the lack of input knowledge and the unavailability of additional DEFAULT-LAYER SBox to be faulted.

To recover the unknown key k_0 , we target the last round of the DEFAULT-CORE and introduce faults individually to each SBox. This technique enables the unique retrieval of the key k_0 . Once an equivalent key is determined, the original key can be obtained by applying the DFA to the DEFAULT-CORE.

To minimize the number of required faults, an efficient strategy involves injecting 8 faults at the fifth-to-last round, allowing the unique determination of \hat{k}_r and k_{r-1} . This strategy is repeated iteratively until only three rounds remain. At this point, injecting 32 faults at the initial round of DEFAULT-LAYER facilitates the unique recovery of \hat{k}_3 and \hat{k}_2 . Finally, injecting two faults at each nibble in the initial round yields the unique choice of k_1 . Subsequently, the DFA is applied to the DEFAULT-CORE to uniquely retrieve k_0 .

3.3 Experimental Results on DEFAULT Under DFA

In this attack scenario, we have conducted a comprehensive analysis for both the simple key schedule and the rotating key schedule of DEFAULT. For the simple key schedule, our estimations indicate that approximately 32, 34, 16, and 5 bit-faults are required to effectively reduce the keyspaces to 2^{32} , 1, 1, and 1, respectively, under a differential fault attack (DFA). These faults are introduced at the second-to-last, third-to-last, fourth-to-last, and fifth-to-last rounds, respectively. Likewise, for the rotating key schedule, our estimates suggest that approximately 96, 48, and 36 bit-faults are necessary to recover the equivalent key for the DEFAULT-LAYER using DFA techniques when the faults are injected at the third-to-last, fourth-to-last, and fifth-to-last rounds, respectively. We have computed the equivalent round keys for the DEFAULT-LAYER and determined that around 32 bit-faults at each SBox in the second-to-last round are sufficient to uniquely recover the key of DEFAULT-CORE. It is important to emphasize that all our

findings and estimations have undergone rigorous practical experiments to ensure their validity and reliability. Detailed implementations of these attacks can be found in [1]. Our experiments were conducted on an Intel® Core™ i5-8250U computer. It is worth noting that employing more powerful computing hardware could potentially yield more accurate fault estimation results.

4 Introducing SDFA: Statistical-Differential Fault Attack on DEFAULT Cipher

In addition to Difference-based Fault Analysis (DFA [10]), Statistical Fault Attack (SFA [16]) is another powerful attack in the context of fault attacks and their analysis. SFA leverages the statistical bias introduced by injected faults and differs from previous attacks is that it only requires faulty ciphertexts, making it applicable in various scenarios compared to difference-based fault attacks. While the designers of the DEFAULT cipher claim that their proposed design can protect against DFA and any form of difference-based fault attacks, but they do not assert security against other fault attacks that exploit statistical biases in the execution. In such scenarios, the designers recommend for the adoption of specialized countermeasures designed to thwart Statistical Ineffective Fault Analysis (SIFA) [13, 14] and Fault Template Attack (FTA) [11, 29]. These countermeasures are recommended to mitigate the inherent risks associated with these specific types of attacks.

Although countermeasures against statistical ineffective fault attacks and fault template attacks can enhance the resilience of a cryptographic system, the absence of specific countermeasures against difference-based fault attacks leaves a potential vulnerability to bit-set faults. Bit-set faults involve intentional manipulations of individual or groups of bits, allowing attackers to strategically modify intermediate values or ciphertexts. Practical experiments [22, 26] on a microcontroller demonstrated successful induction of bit-set faults using laser beams, with higher occurrence rates than bit-flip faults. Despite requiring expensive equipment, this method allows for precise fault injection in target location and timing, as shown in [32]. Without targeted countermeasures against difference-based fault attacks exploiting the propagation of differences through the algorithm, bit-set faults pose a potential risk of revealing sensitive information or compromising system security.

In this section, we introduce a new fault attack called SDFA, which combines DFA with SFA by inducing bit-set faults. The SDFA attack enables us to further reduce the number of faults required to recover the key compared to our proposed improved attacks for both simple and rotating key schedules. Additionally, we demonstrate the effectiveness of this attack in retrieving subkeys for rotating key schedules, even when all the subkeys are generated from a random source.

4.1 Learned Information via SDFA

In Sect. 2.4, we discussed the information learned from DFA and its relation to input-output differences in an SBox. In this section, we delve deeper into the

connection between DFA and SFA when bit-set faults are introduced into the state. Specifically, we examine the scenario where four bit-set faults are applied to positions in the last round SBox, resulting in the unique recovery of the key nibble using SFA. Alternatively, by introducing a bit-set fault in a nibble, we can narrow down the key nibble space from 2^4 to 2^{4-t} , $1 \leq t \leq 2$. Our objective is to combine the power of SFA and DFA to uniquely recover the key nibble with fewer faults in a nibble.

Consider an SBox with inputs (u_0, u_1, u_2, u_3) and outputs (v_0, v_1, v_2, v_3) . Given an input-output difference $\alpha \rightarrow \beta$ in the SBox, the set of possible output nibbles that satisfy the given differential can be represented as $\mathcal{D}_i \cup \mathcal{D}_j$, where $i, j \in \{0, 1, 2, 3\}$ (for the definition of \mathcal{D}_i 's, please refer to Sect. 2.4). Now, let us assume an attacker injects a bit-set fault at the 0-th bit of the SBox, resulting in $u_0 = 1$, and the input difference $\alpha = 1$. Depending on the DDT table, this leads to either $\beta = 3$ or $\beta = 9$. Consequently, the set (\mathcal{D}) of outputs that satisfy the differential $\alpha \rightarrow \beta$ will be either $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_3$ for $\beta = 3$, or $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$ for $\beta = 9$. Simultaneously, for SFA, the attacker can compute the set of outputs \mathcal{I} that satisfy $u_i = 1$ by inverting the SBox using the faulty outputs, i.e., $\mathcal{I} = \{x : S^{-1}(x) \& 2^i = 2^i\}$.

To determine the intersecting nibbles between DFA and SFA, our objective is to identify the common nibble values from each of the four partition sets \mathcal{D}_i for DFA. These sets are denoted as \mathcal{H}_i and defined as $\mathcal{H}_i = \{x \in \mathcal{D} : S^{-1}(x) \& 2^i = 2^i\}$. The table below provides the sets \mathcal{H}_i corresponding to different bit-sets at the i^{th} position. These sets \mathcal{H}_i are obtained by identifying the common values found within the intersecting sets of \mathcal{D} for DFA and \mathcal{I} for SFA.

Bit-Set	\mathcal{H}_0	\mathcal{H}_1	\mathcal{H}_2	\mathcal{H}_3
$u_0 = 1$	$\{5, f\}$	$\{4, e\}$	$\{2, 8\}$	$\{3, 9\}$
$u_1 = 1$	$\{5, a\}$	$\{1, e\}$	$\{7, 8\}$	$\{6, 9\}$
$u_2 = 1$	$\{5, a\}$	$\{4, b\}$	$\{2, d\}$	$\{6, 9\}$
$u_3 = 1$	$\{5, f\}$	$\{1, b\}$	$\{2, 8\}$	$\{6, c\}$

Finally, for each bit-set u_i in the SBox, if $\mathcal{D} = \mathcal{D}_p \cup \mathcal{D}_q, p, q \in \{0, \dots, 3\}$ represents the set of outputs that satisfy the differential $\alpha \rightarrow \beta$, then the SDFA (Statistical-Differential Fault Attack) is defined as the set \mathcal{Z} of possible outputs that satisfy the differential $\alpha \rightarrow \beta$, given by $\mathcal{Z} = \mathcal{D} \cap \mathcal{I} = \mathcal{H}_p \cup \mathcal{H}_q$. An example of the intersecting outputs obtained by performing SDFA under a bit-set fault at the second bit position in the SBox is presented in Example 1.

Now consider a toy cipher where given a message m , the ciphertext c is produced by $c = S(m) \oplus k$. From the above example, the attacker can learn the following two independent equations involving the key bits as follows:

$$\begin{aligned}
 k_0 \oplus k_2 &= (c_0 \oplus c_2) \oplus (v_0 \oplus v_2) = c_0 \oplus c_2, \\
 k_2 \oplus k_3 &= (c_2 \oplus c_3) \oplus (v_2 \oplus v_3) = c_2 \oplus c_3 \oplus 1.
 \end{aligned}$$

Likewise, for any S-box differential $\alpha \rightarrow \beta$ involving bit-sets in the SBox, the attacker can extract two independent equations that involve the key bits, thereby revealing two bits of information about that key nibble. Table ?? provides a comprehensive list of possible differentials under nibble bit-sets, along with their

corresponding independent equations that can be derived through the SDFA attack. It is important to note that in the case of bit-set faults, if the targeted bit is already set to 1, no difference will be generated. In such cases, the DFA attack cannot be performed. However, the SFA attack can still be applied to reduce the key information by one bit. Therefore, even if bit-set faults fail to generate a difference, they can still contribute to the reduction of one key bit information.

Example 1. Let us consider the input-output difference $2 \rightarrow 7$ corresponding to the bit-set $u_1 = 1$ in an S-box. In this case, the set \mathcal{D} of output differences corresponding to the DFA will be $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_2 = \{0, 5, a, f, 2, 7, 8, d\}$. Similarly, for SFA, the set \mathcal{I} will be $\mathcal{I} = \{1, 5, 6, 7, 8, 9, a, e\}$. Therefore, the intersecting set \mathcal{Z} is obtained as $\mathcal{Z} = \mathcal{D} \cap \mathcal{I} = \{5, a, 7, 8\}$. Alternatively, we can compute $\mathcal{H}_0 = \{5, a\}$ and $\mathcal{H}_2 = \{7, 8\}$, which are the sets of output differences in \mathcal{D} that satisfy the condition $(S^{-1}(x) \& 2^i) = 2^i$. Then, the set \mathcal{Z} can be expressed as $\mathcal{Z} = \mathcal{H}_0 \cup \mathcal{H}_2 = \{5, a, 7, 8\}$.

4.2 Attack on Simple Key Schedule

We observe that a single bit-set at the SBox can effectively extract at most two bits of information from the key nibble. Additionally, from the insights provided in table below, we observe that any two bit-sets at the SBox can reduce at most four bits of information, i.e., to generate four independent equations involving the key bits.

Direction	Learned Expression							
	$u_0 = 1$		$u_1 = 1$		$u_2 = 1$		$u_3 = 1$	
	$1 \rightarrow 3$	$1 \rightarrow 9$	$2 \rightarrow 7$	$2 \rightarrow d$	$4 \rightarrow 7$	$4 \rightarrow d$	$8 \rightarrow 6$	$8 \rightarrow c$
Enc (S^{-1})	$\sum_{i=0}^3 k_i$ $k_1 \oplus k_2 \oplus k_3$	$\sum_{i=0}^3 k_i$ k_0	$k_0 \oplus k_2$ $k_2 \oplus k_3$	$k_0 \oplus k_1$ $k_1 \oplus k_2 \oplus k_3$	$k_0 \oplus k_1$ $k_0 \oplus k_2$	$k_0 \oplus k_2$ $k_0 \oplus k_3$	k_0 $k_1 \oplus k_3$	k_0 $k_1 \oplus k_3$

This enables us to uniquely recover the key nibble. In the worst case, it can reduce atleast two bits of information for two bit-sets in a nibble.

If our focus is on the last round of the DEFAULT-LAYER, in the best case scenario we can achieve the unique recovery of each key nibble by injecting 2 faults (active bit-set faults). In the worst case, 4 bit-set faults ensure the unique key recovery of each key nibbles. This shows that around 64 active bit-set faults (in the best case) are required to retrieve the key uniquely. Whereas in the worst case scenario 128 active bit-set faults are sufficient to recover the key. However, to minimize the number of faults required, the attacker can strategically inject bit-set faults in the upper rounds.

4.3 Attack on Rotating Key Schedule

The rotating key schedule in DEFAULT-LAYER involves four keys, namely $k_0, k_1, k_2,$ and $k_3,$ which are used for each round in a rotating fashion. The master key k_0 serves as the initial key, and the other three keys are derived by applying the four

unkeyed round function of DEFAULT-LAYER recursively. From the perspective of an attacker, if any one of the round keys is successfully recovered, it becomes possible to derive the remaining three keys using the key schedule function. In the case of DEFAULT-LAYER, the key k_3 is used in the last round. By injecting approximately three bit-set faults at each nibble in the last round, it is feasible to effectively retrieve the key k_3 .

To summarize, a total of around 64 to 128 faults are required to recover the complete set of keys in DEFAULT-LAYER. This attack strategy leverages the relationship between the round keys and the rotating key schedule, allowing for the recovery of the master key and subsequent derivation of the other keys.

4.4 Generic Attack on Truly Independent Random Keys

In the scenario where the round keys in the DEFAULT cipher are genuinely generated from random sources rather than derived from a master key using recursive unkeyed round functions, the task of uniquely retrieving all the keys becomes considerably more challenging. In this case, both our DFA approach and the strategy presented in [24] face significant challenges in recovering keys uniquely and may require injecting a substantially larger number of faults compared to our SDFA approach.

Simply speaking, the SDFA approach involves injecting approximately three bit-set faults at each round of DEFAULT-LAYER and utilizing these faults to achieve unique key recovery. Thus, when the round keys are genuinely independent and not derived from a master key, this strategy proves to be much more effective than the DFA strategy. To provide a more concrete perspective, if DEFAULT employs a total of x ($x > 29$) truly independent round keys, then approximately $x \times y$, $y \in [64, 128]$ bit-set faults are needed to recover all of its independent keys. This substantial increase in the number of required faults underscores the heightened difficulty of retrieving the keys when they are genuinely independent and not derived from a common source.

4.5 Experimental Results on DEFAULT Under SDFA

We have performed an extensive analysis utilizing our novel attack strategy, SDFA, on both the simple key schedule and the rotating key schedule, considering the bit-set fault scenario. In the most favorable scenario for both key schedules, our estimations indicate that 64 active bit-set faults, with two faults introduced at each SBox, are adequate to uniquely recover the encryption key. Conversely, in the most challenging scenario, injecting 128 active bit-set faults at each SBox guarantees the unique key recovery. For complete implementation details of these attacks, we refer to [1]. The experiment was conducted on an Intel® Coretm i5-8250U computer.

5 Attacks on BAKSHEESH

For the BAKSHEESH cipher, despite the absence of any claimed DFA security by the designer, we conducted a thorough examination of its susceptibility to

both Differential Fault Analysis (DFA) and Statistical-Differential Fault Analysis (SDFA) under bit-flip and bit-set fault scenarios, respectively. In this section, we will begin by outlining the differential fault attack, wherein we introduce faults at various rounds and determine the minimum number of faults required to achieve unique key recovery. Subsequently, we will present the SDFA attack and provide an estimate of the number of faults necessary to successfully retrieve the key in a unique manner.

5.1 DFA on BAKSHEESH

In this section, we outline our strategy for efficiently determining the differential trail up to three rounds to facilitate DFA attacks. We explain the trail computation process, its application in key retrieval via bit-flip faults, and estimate the fault complexity for key recovery in various rounds.

5.1.1 Faults at the Last Round

In our observations, injecting two faults at each nibble in the last round of BAKSHEESH yields three bits of information. Additionally, it is worth noting that the two key values corresponding to any two injected faults at the SBox are complementary to each other. The initial approach to reduce the key space involves inducing two bit-flip faults at each nibble in the last round before the SBox operation, individually affecting key nibbles, thus reducing the key space to 2^{32} with 64 faults in the last round. However, a more efficient strategy is required, inducing faults further from the last rounds, and deterministically obtaining information about the input differences for each SBox in the last round. This necessitates the development of a deterministic strategy capable of guessing the differential path from which the faults originate. In the upcoming subsections, we will demonstrate the feasibility of deterministically computing the differential path in BAKSHEESH for up to three rounds.

5.1.2 Faults at the Second-to-Last Round

The GIFT-128 permutation structure of the cipher permits a nibble difference at the input of group \mathcal{G}_{r_i} in the second-to-last round to induce a bit difference in four nibbles in the last round. This observation allows an attacker to deterministically ascertain the differential path by introducing bit-flip faults at the second-to-last round. Furthermore, this insight enables the deterministic computation of differential paths for up to three rounds, as discussed in the next subsection. This is achievable because, for both non-faulty and faulty ciphertexts, the last round can be inverted by assessing input bit-differences at each nibble using DDT. The internal state difference can then be calculated by examining input bit-differences after the inverse operation of the second-to-last round, leveraging the Quotient-Remainder group structure.

A straightforward approach to attacking the cipher involves injecting two bit-faults at each nibble in the last round, thereby reducing the key space for each

nibble to 2, resulting in an overall keyspace of 2^{32} . Subsequently, injecting one fault at each nibble in the second-to-last round uniquely reduces the keyspace. This naive approach necessitates approximately 96 faults for key recovery. However, we can enhance this attack by introducing faults at the second-to-last round during encryption. Our practical validation confirms that the introduction of one bit-faults at the second bit position in each SBox and two bit-faults at the third bit position in two different SBox at each group \mathcal{G}_i at the second-to-last round, substantially diminishes the keyspace to nearly unique key. Detailed information on the reduced keyspace values corresponding to different fault injection counts is available in Table 2.

5.1.3 Faults at the Third-to-Last Round

In this attack, we introduce bit-faults into a nibble during the third-to-last round of the cipher. Similar to the previous attack in DEFAULT-LAYER, we follow a deterministic process to calculate the input and output differences for each nibble at every round. This allows us to track how differences propagate throughout the cipher, as illustrated in Fig. 2. Also, the three rounds trail computation is similar to Algorithm 1. We then leverage the computed trail to reduce the cipher's keyspace. By introducing two distinct bit differences in each nibble during the last round, we effectively reduce the keyspace to 2^{32} . Next, our focus narrows down to nibble positions 0, 1, 2, 3, 8, 9, 10, and 11 during the second-to-last round. We filter these nibble positions by iteratively inverting two rounds relative to combining the keyspaces from nibble positions 0, 1, 2, 3, 8, 9, 10, 11, 16, 17, 18, 19, 24, 25, 26, and 27, all based on the key nibbles of the last round. Similarly, we filter nibble positions 20, 21, 22, 23, 28, 29, 30, and 31 by inverting two rounds with respect to combining the keyspaces from nibble positions 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, and 31, again based on the key nibbles of the last round. We subsequently perform further filtering on remaining nibble differences at the second-to-last round, considering the reduced keyspace for all 32 key nibble positions. Finally, we conduct additional filtering on nibble differences at the third-to-last round based on the further reduced keyspace. Our practical verification demonstrates that introducing two bit-faults at the third bit position in two different SBox within each group \mathcal{G}_i during the third-to-last round significantly reduces the keyspace to a unique key. Comprehensive details regarding the reduced keyspace values for various fault injection counts can be found in Table 2.

5.2 SDFA on BAKSHEESH

The SBox employed in the BAKSHEESH cipher features a single non-zero LS element, denoted as 8. In the context of DFA, the key nibbles can be effectively reduced to one bit by introducing a minimum of two faults in each nibble. Notably, only introducing any two out of the three possible input differences (1, 2, and 4) at each SBox is sufficient to reduce the key nibbles to 2, given that 8 is a LS point.

Regarding SFA, our observations indicate that performing four SFA operations using bit-set faults can reduce the key nibbles to a minimum of 2. We have verified that introducing bit-set faults at each position within the SBox nibbles, with one active fault at the first three positions, is capable of uniquely reducing the key nibble space. Therefore, approximately 128 bit-set faults are sufficient for a nearly unique key recovery.

5.3 Experimental Results on BAKSHEESH

In this attack scenario, we have applied both our DFA and SDFA attack techniques to BAKSHEESH, achieving successful key recovery. In the DFA approach, our estimations suggest that approximately 48 and 16 bit-faults are needed to reduce the keyspaces to $2^{0.2}$ and 1, respectively. These faults are strategically introduced at the second-to-last and third-to-last rounds. When it comes to the SDFA approach, our most favorable estimations indicate that 96 active bit-set faults, with three faults introduced at each SBox, are sufficient for a unique key recovery. In the worst-case scenario, injecting 128 active bit-set faults at each SBox guarantees a unique key recovery. For detailed implementations of these attacks, we refer to [1]. The experiments were performed on an Intel® Core™ i5-8250U computer. It is important to mention that employing more powerful computing hardware could potentially lead to more precise fault estimation results.

6 Discussion

This work presents enhanced DFA attacks on both LS SBox-based ciphers, DEFAULT and BAKSHEESH. The DEFAULT-LAYER SBox incorporates three non-trivial LS elements, while BAKSHEESH has only one non-trivial LS element.

One of the work related to this, is done by Dey et al. [12]. Here the authors inject faults across multiple rounds, namely, in the last and the second-to-last rounds, and use a nibble-based fault model. They inject 64 random nibble faults in the last round to reduce the key space to at least 2^{64} . Further reduction of the key space to 2^{16} is possible only by injecting additional 48 faults in the second-to-last round, yielding a total of 112 faults for a two-round attack. In contrast, our new approach extends the number of rounds, significantly reducing the number of faults compared to [12]. Moreover, their work is on the older version of DEFAULT and does not work on newer versions of the cipher, while ours is applicable to the latest version with rotating key-schedule as well.

Nageler et al. [24] presents another interesting strategy to reduce the key space and mount an attack on 3 rounds of DEFAULT. They utilise a top-to-bottom approach for constructing a collection of possible intermediate state difference lists while simultaneously calculating their probabilities. However, these probabilities are not directly exploited in the key recovery analysis, which slows down the rate of key space reduction across rounds. Our approach, on the other hand, adopts a

bottom-top methodology. The linear layer in DEFAULT is a bit-based permutation from GIFT, allowing for deterministic capture of difference propagation when a fault is injected in the third-to-last encryption round. By computing the differential trail deterministically up to 3 rounds and extending it to 5 rounds through guess-and-determine strategy, we find that each active S-box has a unique input-output difference at all intermediate rounds, yielding a more efficient keyspace reduction. Consequently, our approach requires fewer faults to achieve unique key recovery and to bring the keyspace below a specified threshold. Though our work uses the same fault model as [24], our 3 round attack remains effective under random nibble faults also, both in position and value.

Moreover, the work [24] recovers around 112 key bits with 16 faults. When we use their algorithm to increase the number of faults, we find that their method does not recover beyond 112 key bits. On the other hand, using our technique with only 5 faults, we could reduce the keyspace to identity.

We also study the trade-off between the number of faults and keyspace reduction and compare the corresponding theory with experiments. We find that with 4 and 3 faults, the keyspace size reduces to approximately 2^1 and 2^8 respectively. One single bit input-difference at the fifth-to-last round of DEFAULT SBox can propagate to the output-differences with of hamming weight (\mathcal{HW}) 2 or 3. Assuming an output-difference with $\mathcal{HW}(2)$, we get approximately 16 input-output differences using each of the 4 faults, i.e., 64 in total, thereby reducing keyspace to 2^{64} . In practice, each nibble may not always get at least 2 distinct input-output differences, and thus the keyspace may not be reduced exactly to 2^{64} , but only approximately. Proceeding further, if we combine this with the distinguishers of the lower rounds, the final keyspace reduces to 2^{12} theoretically. However, in practice, we observe 2^1 possibly owing to some output differences having larger hamming weights. With less faults, the product keyspace of 4 remainder groups in the last round becomes much larger, leading to infeasible time complexity.

7 Conclusion

In light of the practical significance of Differential Fault Analysis (DFA) style attacks, the development of effective cipher protection strategies holds substantial relevance. Over recent years, various approaches and strategies have been explored to mitigate such vulnerabilities. Notably, the authors of the DEFAULT cipher have introduced a compelling design strategy aimed at intrinsically constraining the extent of information accessible to potential attackers. The designers claimed its DFA security to be 2^{64} under any difference-based fault analysis.

In this study, we present an enhanced DFA on the DEFAULT cipher, enabling the effective and unique retrieval of the encryption key. Our approach involves constructing deterministic differential trails spanning up to five rounds and applying DFA by injecting faults at various rounds while quantifying the required number of faults. Specifically, for the simple key schedule, we demonstrate that approximately 5 bit-flip faults are sufficient to uniquely recover the key

of DEFAULT. In contrast, for systems utilizing rotating keys, we show that approximately 36 bit-flip faults are required to recover the equivalent key of DEFAULT-LAYER. Remarkably, our attack achieves key recovery with a significantly reduced number of faults compared to previous methods.

When we inject faults at the sixth-to-last round to extend our trail computation techniques to one more round, the differential trails found are not unique. This might slow down the key recovery due to multiple input-output differences at an SBox in the intermediate rounds. The detailed investigation of deeper rounds and keyspace reduction may be an interesting future work.

Furthermore, we introduce a novel fault attack technique known as the Statistical-Differential Fault Attack (SDFA), which combines elements of both Statistical Fault Analysis (SFA) and DFA. In this attack, we demonstrate that at most 128 bit-set faults are sufficient to recover the key for both the key schedule configurations of the DEFAULT cipher. This attack highlights its efficacy in recovering encryption keys, not only for systems employing rotating keys but also for ciphers utilizing entirely round-independent keys.

Finally, we apply our proposed DFA attack to another linear-structured SBox-based cipher, BAKSHEESH, and efficiently recovered its master key uniquely. We show that approximately 16 bit-faults are required to achieve unique key recovery for BAKSHEESH. Similarly, under the bit-set fault model, the SDFA attack can be effectively applied to nearly retrieve its key uniquely by inducing 128 bit-set faults in the worst case.

In conclusion, our work makes significant contributions to the field of fault attacks by presenting enhanced DFA techniques, extending their applicability to rotating and round-independent keys, and introducing the novel SDFA approach to combine the advantages of both DFA and SFA. Our findings underscore the difficulty in achieving DFA protection for linear-structured SBox-based ciphers. In principle, our SDFA can also be applied to any non-linear SBox-based ciphers, which may be an interesting future work.

A Appendix

Algorithm 2. DETERMINISTIC COMPUTATION OF FIVE ROUNDS DIFFERENTIAL TRAIL

Input: A list of ciphertext difference $\mathcal{L}_{\Delta C}$
Output: Lists of input-output differences $\mathcal{A}_{ID}^{23}, \mathcal{A}_{ID}^{24}, \mathcal{A}_{ID}^{25}, \mathcal{A}_{ID}^{26}$, & \mathcal{A}_{ID}^{27}

- 1: $\mathcal{L}_1 \leftarrow [], \mathcal{L}_2 \leftarrow [], \mathcal{A}_{ID}^{23} \leftarrow [[], \mathcal{A}_{ID}^{24} \leftarrow [[], []], \mathcal{A}_{ID}^{25} \leftarrow [[], []], \mathcal{A}_{ID}^{26} \leftarrow [[], []], \mathcal{A}_{ID}^{27} \leftarrow [[], []]$
- 2: $T_1 = [0, 1, 4, 5], T_2 = [0, 2, 8, 10]$
- 3: $\mathcal{T} = [[(T_1)^8, (T_2)^8], (T_1)^8, (T_2)^8], [(T_2)^8, (T_1)^8], [(T_2)^8, (T_1)^8], (T_1)^8] \triangleright$ Input nibble differences at the second-to-last round correspond to faults at the left/right half
- 4: $\mathcal{L}_1 = \mathcal{L}_{\Delta C}$
- 5: $\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$ \triangleright Invert through bit-permutation layer
- 6: **for** $i = 0$ to 31 **do** \triangleright At the round R^{27}
- 7: $\mathcal{A}_{ID}^{27}[1][i] = \mathcal{L}_1[i]$
- 8: **for** $j = 0$ to 1 **do** \triangleright For each fault at the left/right half in the fifth-to-last round
- 9: **for** $i = 0$ to 8 **do** \triangleright For each group $\mathcal{G}r_i$ at R^{26}
- 10: **for** $(\Delta_0, \Delta_1, \Delta_2, \Delta_3) \in S^{-1}(\mathcal{L}_1[i]) \times S^{-1}(\mathcal{L}_1[i+8]) \times S^{-1}(\mathcal{L}_1[i+16]) \times S^{-1}(\mathcal{L}_1[i+24])$
at round R^{27} **do**
- 11: $\mathcal{L}_1[i] = \Delta_0, \mathcal{L}_1[i+8] = \Delta_1, \mathcal{L}_1[i+16] = \Delta_2, \mathcal{L}_1[i+24] = \Delta_3$
- 12: $\mathcal{L}_1[j] = 0, j \notin \{i, i+8, i+16, i+24\}$
- 13: $\mathcal{A}_{ID}^{27}[0] = \mathcal{L}_1$
- 14: $\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$
- 15: $\mathcal{A}_{ID}^{26}[1] = \mathcal{L}_1$
- 16: **for** $(\Delta_0, \Delta_1, \Delta_2, \Delta_3) \in S^{-1}(\mathcal{L}_1[0+\alpha]) \times S^{-1}(\mathcal{L}_1[1+\alpha]) \times S^{-1}(\mathcal{L}_1[2+\alpha]) \times S^{-1}(\mathcal{L}_1[3+\alpha])$
 $\alpha)$ at round R^{26} **do** $\triangleright \alpha \leftarrow 4 * i$, here $(\Delta_0, \Delta_1, \Delta_2, \Delta_3)$ denotes the difference in i -th quotient group
- 17: $\mathcal{L}_2[\alpha] = \Delta_0, \mathcal{L}_2[1+\alpha] = \Delta_1, \mathcal{L}_2[2+\alpha] = \Delta_2, \mathcal{L}_2[3+\alpha] = \Delta_3$
- 18: $\mathcal{L}_2[j] = 0, j \notin \{\alpha, 1+\alpha, 2+\alpha, 3+\alpha\}$
- 19: $\mathcal{A}_{ID}^{26}[0] = \mathcal{L}_2$
- 20: **if** $(\Delta_0 \in \mathcal{T}[j][\alpha]) \& (\Delta_1 \in \mathcal{T}[j][1+\alpha]) \& (\Delta_2 \in \mathcal{T}[j][2+\alpha]) \& (\Delta_3 \in \mathcal{T}[j][3+\alpha])$
then
- 21: $\mathcal{L}_{\Delta C} = \mathcal{L}_2$
- 22: Compute the trail for other three rounds using Algorithm 1 and get $\mathcal{A}_{ID}^{25}, \mathcal{A}_{ID}^{24}$, and \mathcal{A}_{ID}^{23}
- 23: **return** the lists $\mathcal{A}_{ID}^{27}, \mathcal{A}_{ID}^{26}, \mathcal{A}_{ID}^{25}, \mathcal{A}_{ID}^{24}$ and \mathcal{A}_{ID}^{23}

Algorithm 3. DETERMINISTIC COMPUTATION OF FOUR ROUNDS DIFFERENTIAL TRAIL

Input: A list of ciphertext difference $\mathcal{L}_{\Delta C}$
Output: Lists of input-output differences $\mathcal{A}_{ID}^{24}, \mathcal{A}_{ID}^{25}, \mathcal{A}_{ID}^{26}$, & \mathcal{A}_{ID}^{27}

- 1: Initialize $\mathcal{L}_1 \leftarrow [], \mathcal{A}_{ID}^{24} \leftarrow [[], []], \mathcal{A}_{ID}^{25} \leftarrow [[], []], \mathcal{A}_{ID}^{26} \leftarrow [[], []], \mathcal{A}_{ID}^{27} \leftarrow [[], []]$
- 2: $\mathcal{L}_1 = \mathcal{L}_{\Delta C}$
- 3: $\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$ \triangleright Invert through bit-permutation layer
- 4: **for** $i = 0$ to 31 **do** \triangleright At the round R^{27}
- 5: $\mathcal{A}_{ID}^{27}[1][i] = \mathcal{L}_1[i]$
- 6: **for** $i = 0$ to 8 **do** \triangleright For each group $\mathcal{G}r_i$ at R^{26}
- 7: **for** $(\Delta_0, \Delta_1, \Delta_2, \Delta_3) \in S^{-1}(\mathcal{L}_1[i]) \times S^{-1}(\mathcal{L}_1[i+8]) \times S^{-1}(\mathcal{L}_1[i+16]) \times S^{-1}(\mathcal{L}_1[i+24])$
at round R^{27} **do**
- 8: $\mathcal{L}_1[i] = \Delta_0, \mathcal{L}_1[i+8] = \Delta_1, \mathcal{L}_1[i+16] = \Delta_2, \mathcal{L}_1[i+24] = \Delta_3$
- 9: $\mathcal{L}_1[j] = 0, j \notin \{i, i+8, i+16, i+24\}$
- 10: $\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$
- 11: **if** $\mathcal{L}_1[j] = 0, \forall j \in \{0, \dots, 31\} \setminus \{\alpha, \alpha+1, \alpha+2, \alpha+3\}$ **then** $\triangleright \alpha \leftarrow 4 * i$
- 12: **if** $j \in \{0, 1\}$ **then** $\triangleright j = 0/1 \rightarrow$ injected faults at the left/right half of R^{24}
- 13: **if** $S^{-1}(\mathcal{L}_1[\alpha+j]) \notin \mathcal{S}$ or $S^{-1}(\mathcal{L}_1[\alpha+j+2]) \notin \mathcal{S}$ **then** $\triangleright \mathcal{S} \leftarrow \{1, 2, 4, 8\}$
- 14: Break the for loop
- 15: $\mathcal{A}_{ID}^{27}[0][i] = \Delta_0, \mathcal{A}_{ID}^{27}[0][i+8] = \Delta_1, \mathcal{A}_{ID}^{27}[0][i+16] = \Delta_2, \mathcal{A}_{ID}^{27}[0][i+24] = \Delta_3$
- 16: $\mathcal{L}_{\Delta C}[i] = \Delta_0, \mathcal{L}_{\Delta C}[i+8] = \Delta_1, \mathcal{L}_{\Delta C}[i+16] = \Delta_2, \mathcal{L}_{\Delta C}[i+24] = \Delta_3$
- 17: Compute the trail for other three rounds using Algorithm 1 and get $\mathcal{A}_{ID}^{26}, \mathcal{A}_{ID}^{25}$ and \mathcal{A}_{ID}^{24}
- 18: **return** the lists $\mathcal{A}_{ID}^{27}, \mathcal{A}_{ID}^{26}, \mathcal{A}_{ID}^{25}$ and \mathcal{A}_{ID}^{24}

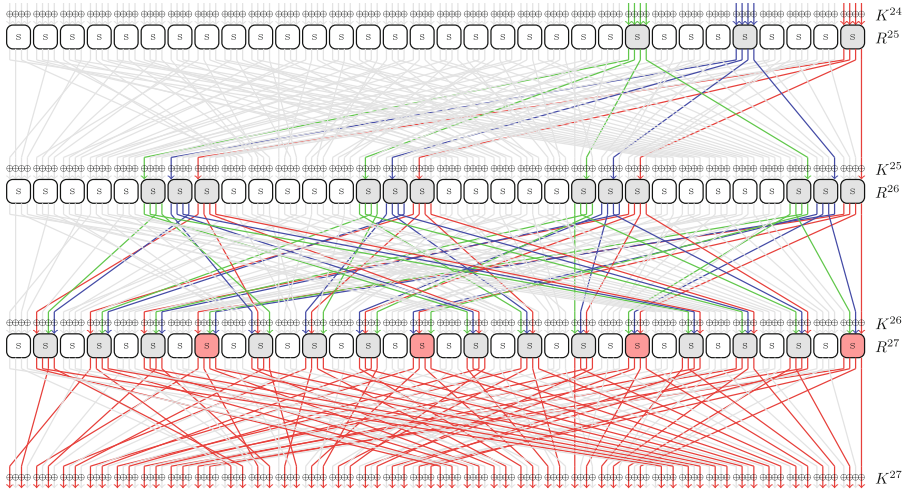


Fig. 2. Fault Propagation for Three Rounds

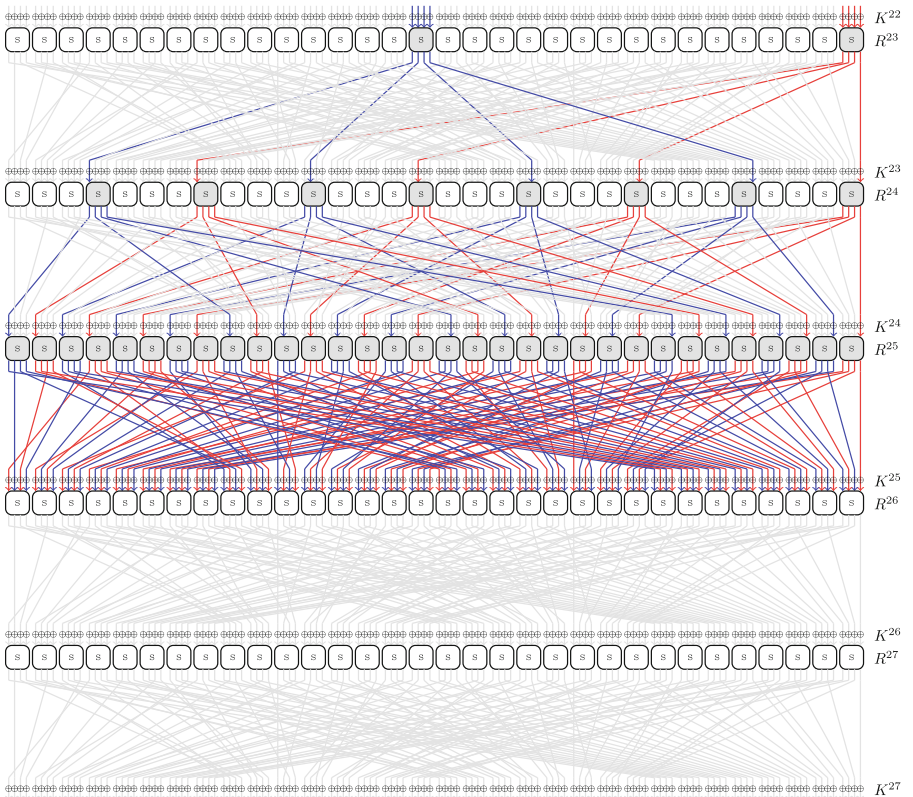


Fig. 3. Fault Propagation for Five Rounds

References

1. Attack Verification of DEFAULT and BAKSHEESH. <https://github.com/anup557/sdfa>
2. Agoyan, M., Dutertre, J., Mirbaha, A., Naccache, D., Ribotta, A., Tria, A.: How to flip a bit? In: 16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece. pp. 235–239. IEEE Computer Society (2010), <https://doi.org/10.1109/IOLTS.2010.5560194>
3. Baksi, A., Bhasin, S., Breier, J., Jap, D., Saha, D.: A survey on fault attacks on symmetric key cryptosystems. *ACM Comput. Surv.* **55**(4), 86:1–86:34 (2023). <https://doi.org/10.1145/3530054>
4. Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T., Sarkar, S., Sim, S.M.: DEFAULT: cipher level resistance against differential fault attack. In: Tibouchi, M., Wang, H. (eds.) *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security*, Singapore, December 6-10, 2021, Proceedings, Part II. *Lecture Notes in Computer Science*, vol. 13091, pp. 124–156. Springer (2021). https://doi.org/10.1007/978-3-030-92075-3_5
5. Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T., Sarkar, S., Sim, S.M.: DEFAULT: cipher level resistance against differential fault attack. *IACR Cryptol. ePrint Arch.* p. 712 (2021), <https://eprint.iacr.org/archive/2021/712/1622193888.pdf>
6. Baksi, A., Breier, J., Chattopadhyay, A., Gerlich, T., Guilley, S., Gupta, N., Hu, K., Isobe, T., Jati, A., Jedlicka, P., Kim, H., Liu, F., Martinasek, Z., Sakamoto, K., Seo, H., Shiba, R., Shrivastwa, R.R.: BAKSHEESH: similar yet different from GIFT. *IACR Cryptol. ePrint Arch.* p. 750 (2023), <https://eprint.iacr.org/2023/750>
7. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: A small present - towards reaching the limit of lightweight encryption. In: Fischer, W., Homma, N. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference*, Taipei, Taiwan, September 25-28, 2017, Proceedings. *Lecture Notes in Computer Science*, vol. 10529, pp. 321–345. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_16
8. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pellicoli, M., Pelosi, G.: Fault attack on AES with single-bit induced faults. In: *Sixth International Conference on Information Assurance and Security, IAS 2010*, Atlanta, GA, USA, August 23-25, 2010. pp. 167–172. IEEE (2010). <https://doi.org/10.1109/ISIAS.2010.5604061>
9. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symmetric Cryptol.* **2019**(1), 5–45 (2019). <https://doi.org/10.13154/tosc.v2019.i1.5-45>
10. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Jr., B.S.K. (ed.) *Advances in Cryptology - CRYPTO '97*, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings. *Lecture Notes in Computer Science*, vol. 1294, pp. 513–525. Springer (1997). <https://doi.org/10.1007/BFb0052259>
11. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2002*, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. *Lecture Notes in Computer Science*, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3

12. Dey, C., Pandey, S.K., Roy, T., Sarkar, S.: Differential fault attack on DEFAULT. *IACR Cryptol. ePrint Arch.* p. 1392 (2021), <https://eprint.iacr.org/2021/1392>
13. Dobraunig, C., Eichlseder, M., Groß, H., Mangard, S., Mendel, F., Primas, R.: Statistical ineffective fault attacks on masked AES with fault countermeasures. In: Peyrin, T., Galbraith, S.D. (eds.) *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II. Lecture Notes in Computer Science*, vol. 11273, pp. 315–342. Springer (2018). https://doi.org/10.1007/978-3-030-03329-3_11
14. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**(3), 547–572 (2018). <https://doi.org/10.13154/tches.v2018.i3.547-572>
15. Dutertre, J., Mirbaha, A., Naccache, D., Ribotta, A., Tria, A., Vaschalde, T.: Fault round modification analysis of the advanced encryption standard. In: 2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012. pp. 140–145. IEEE Computer Society (2012). <https://doi.org/10.1109/HST.2012.6224334>
16. Fuhr, T., Jaulmes, É., Lomné, V., Thillard, A.: Fault attacks on AES with faulty ciphertexts only. In: Fischer, W., Schmidt, J. (eds.) *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*. pp. 108–118. IEEE Computer Society (2013). <https://doi.org/10.1109/FDTC.2013.18>,
17. Jana, A.: Differential fault attack on feistel-based sponge AE schemes. *J. Hardw. Syst. Secur.* **6**(1-2), 1–16 (2022). <https://doi.org/10.1007/s41635-022-00124-w>
18. Jana, A., Paul, G.: Differential fault attack on photon-beetle. In: Chang, C., Rührmair, U., Mukhopadhyay, D., Forte, D. (eds.) *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security, ASHES 2022, Los Angeles, CA, USA, 11 November 2022*. pp. 25–34. ACM (2022). <https://doi.org/10.1145/3560834.3563824>,
19. Jana, A., Saha, D., Paul, G.: Differential fault analysis of NORX. In: Chang, C., Rührmair, U., Katzenbeisser, S., Schaumont, P. (eds.) *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2020, Virtual Event, USA, November 13, 2020*. pp. 67–79. ACM (2020). <https://doi.org/10.1145/3411504.3421213>
20. Kim, Y., Daly, R., Kim, J.S., Fallin, C., Lee, J., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. pp. 361–372. IEEE Computer Society (2014). <https://doi.org/10.1109/ISCA.2014.6853210>,
21. Kundu, A.K., Ghosh, S., Aikata, A., Saha, D.: Tofa: Towards fault analysis of gift and gift-like ciphers leveraging truncated impossible differentials [unpublished manuscript] (2022)
22. Menu, A., Dutertre, J., Rigaud, J., Colombier, B., Moëllic, P., Danger, J.: Single-bit laser fault model in NOR flash memories: Analysis and exploitation. In: *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*. pp. 41–48. IEEE (2020). <https://doi.org/10.1109/FDTC51366.2020.00013>,
23. Moradi, A., Shalmani, M.T.M., Salmasizadeh, M.: A generalized method of differential fault attack against AES cryptosystem. In: Goubin, L., Matsui, M. (eds.)

- Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4249, pp. 91–100. Springer (2006), https://doi.org/10.1007/11894063_8
24. Nageler, M., Dobraunig, C., Eichlseder, M.: Information-combining differential fault attacks on DEFAULT. In: Dunkelman, O., Dziembowski, S. (eds.) *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13277, pp. 168–191. Springer (2022). https://doi.org/10.1007/978-3-031-07082-2_7
 25. Piret, G., Quisquater, J.: A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2003*, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2779, pp. 77–88. Springer (2003), https://doi.org/10.1007/978-3-540-45238-6_7
 26. Roscian, C., Sarafianos, A., Dutertre, J., Tria, A.: Fault model analysis of laser-induced faults in SRAM memory cells. In: Fischer, W., Schmidt, J. (eds.) *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Los Alamitos, CA, USA, August 20, 2013. pp. 89–98. IEEE Computer Society (2013). <https://doi.org/10.1109/FDTC.2013.17>,
 27. Saha, D., Chowdhury, D.R.: Scope: On the side channel vulnerability of releasing unverified plaintexts. In: Dunkelman, O., Keliher, L. (eds.) *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference*, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9566, pp. 417–438. Springer (2015). https://doi.org/10.1007/978-3-319-31301-6_24
 28. Saha, D., Chowdhury, D.R.: Encounter: On breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In: Gierlichs, B., Poschmann, A.Y. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference*, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9813, pp. 581–601. Springer (2016). https://doi.org/10.1007/978-3-662-53140-2_28
 29. Saha, S., Bag, A., Roy, D.B., Patranabis, S., Mukhopadhyay, D.: Fault template attacks on block ciphers exploiting fault propagation. In: Canteaut, A., Ishai, Y. (eds.) *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 612–643. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_22
 30. Selmke, B., Brummer, S., Heyszl, J., Sigl, G.: Precise laser fault injections into 90 nm and 45 nm sram-cells. In: Homma, N., Medwed, M. (eds.) *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015*, Bochum, Germany, November 4-6, 2015. Revised Selected Papers. Lecture Notes in Computer Science, vol. 9514, pp. 193–205. Springer (2015). https://doi.org/10.1007/978-3-319-31271-2_12
 31. Simon, T., Batina, L., Daemen, J., Grosso, V., Massolino, P.M.C., Papagiannopoulos, K., Regazzoni, F., Samwel, N.: Friet: An authenticated encryption scheme with built-in fault detection. In: Canteaut, A., Ishai, Y. (eds.) *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory*

- and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 581–611. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_21
32. Skorobogatov, S.: Optical fault masking attacks. In: Breveglieri, L., Joye, M., Koren, I., Naccache, D., Verbauwhede, I. (eds.) 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010. pp. 23–29. IEEE Computer Society (2010). <https://doi.org/10.1109/FDTC.2010.18>,
 33. Tunstall, M., Mukhopadhyay, D.: Differential fault analysis of the advanced encryption standard using a single fault. IACR Cryptol. ePrint Arch. p. 575 (2009), <http://eprint.iacr.org/2009/575>

Cryptanalysis on Various Problems



Hard-Label Cryptanalytic Extraction of Neural Network Models

Yi Chen¹, Xiaoyang Dong^{2,5}, Jian Guo³, Yantian Shen⁴,
Anyu Wang^{1,5}, and Xiaoyun Wang^{1,5,6}

¹ Institute for Advanced Study, Tsinghua University, Beijing, China
chenyi2023@mail.tsinghua.edu.cn, {anyuwang,xiaoyunwang}@tsinghua.edu.cn

² Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University,
Beijing, China
xiaoyangdong@tsinghua.edu.cn

³ School of Physical and Mathematical Sciences, Nanyang Technological University,
Singapore, Singapore
guojian@ntu.edu.sg

⁴ Department of Computer Science and Technology, Tsinghua University,
Beijing, China
shenyt22@mails.tsinghua.edu.cn

⁵ Zhongguancun Laboratory, Beijing, China

⁶ Shandong Key Laboratory of Artificial Intelligence Security, Shandong, China

Abstract. The machine learning problem of extracting neural network parameters has been proposed for nearly three decades. Functionally equivalent extraction is a crucial goal for research on this problem. When the adversary has access to the raw output of neural networks, various attacks, including those presented at CRYPTO 2020 and EURO-CRYPT 2024, have successfully achieved this goal. However, this goal is not achieved when neural networks operate under a hard-label setting where the raw output is inaccessible.

In this paper, we propose the first attack that theoretically achieves functionally equivalent extraction under the hard-label setting, which applies to ReLU neural networks. The effectiveness of our attack is validated through practical experiments on a wide range of ReLU neural networks, including neural networks trained on two real benchmarking datasets (MNIST, CIFAR10) widely used in computer vision. For a neural network consisting of 10^5 parameters, our attack only requires several hours on a single core.

Keywords: Cryptanalysis · ReLU Neural Networks · Functionally Equivalent Extraction · Hard-Label

1 Introduction

Extracting all the parameters (including weights and biases) of a neural network (called victim model) is a long-standing open problem which is first proposed by

cryptographers and mathematicians in the early nineties of the last century [2, 7], and has been widely studied by research groups from both industry and academia [1, 3, 4, 11, 14, 16, 18, 20].

In previous research [3, 4, 11, 14, 16, 18, 20], one of the most common attack scenarios is as follows. The victim model (denoted by f_θ where θ denotes the parameters) is made available as an Oracle \mathcal{O} , then the adversary generates inputs x to query \mathcal{O} and collects the feedback ζ to extract the parameters. This is similar to a cryptanalysis problem: θ is considered the secret key, and the adversary tries to recover the secret key θ , given the pairs (x, ζ) [4]. If f_θ contains m parameters (64-bit floating-point numbers), then the secret key θ contains $64 \times m$ bits, and the computation complexity of brute force searching is $2^{64 \times m}$.

Consider that there may be isomorphisms for neural networks, e.g., permutation and scaling for ReLU neural networks [18]. An important concept named *functionally equivalent extraction* is summarized and proposed in [11].

Functionally Equivalent Extraction. Denote by \mathcal{X} the input space of the victim model f_θ . Functionally equivalent extraction aims at generating a model $f_{\hat{\theta}}$ (i.e., the extracted model), such that $f_\theta(x) = f_{\hat{\theta}}(x)$ holds for all $x \in \mathcal{X}$, where $f_\theta(x)$ and $f_{\hat{\theta}}(x)$ are, respectively, the raw output of the victim model and the extracted model [11]. Such extracted model $f_{\hat{\theta}}$ is called the functionally equivalent model of f_θ (also say that f_θ and $f_{\hat{\theta}}$ are isomorphic [18]). Since $f_{\hat{\theta}}$ behaves the same as f_θ , the adversary can explore the properties of f_θ by taking $f_{\hat{\theta}}$ as a perfect substitute¹.

Functionally equivalent extraction is hard [11]. Consider the isomorphisms (permutation and scaling) introduced in [18]. Scaling can change parameters and permutation does not. For a ReLU neural network f_θ that contains m parameters (64-bit floating-point numbers) and n neurons, from the perspective of the above cryptanalysis problem, even though *scaling can be applied to each neuron once*, the adversary still needs to recover $64 \times (m - n)$ secret key bits. Note that the case of $m \gg n$ is common for neural networks. For example, for the ReLU neural networks extracted by Carlini et al. at CRYPTO 2020 [4], the pairs (m, n) are (25120, 32), (100480, 128), (210, 20), (420, 40), and (4020, 60). Besides, even if we only recover a few bits (instead of 64 bits) of each parameter, the number of secret key bits to be recovered is still large, particularly in the case of large m .

When a functionally equivalent model $f_{\hat{\theta}}$ is obtained, the adversary can do more damage (e.g., adversarial attack [5]) or steal user privacy (e.g., property inference attack [9]). Thus, even though the cost is expensive, the adversary has the motivation to achieve functionally equivalent extraction.

Hard-Label Setting. According to the taxonomy in [11], when the Oracle is queried, there are 5 types of feedback given by the Oracle: (1) label (the most likely class label, also called hard-label), (2) label and score (the most-likely class

¹ Due to the isomorphisms introduced in [18], the parameters $\hat{\theta}$ of the extracted model may be different from that θ of the victim model, but it does not matter as long as $f_{\hat{\theta}}$ is the functionally equivalent model of f_θ .

label and its probability score), (3) top-k scores, (4) all the label scores, (5) the raw output (i.e., $f_\theta(x)$). When the Oracle only returns the hard-label, we say that the victim model f_θ (i.e., neural networks in this paper) works under the hard-label setting [8].

To the best of our knowledge, there are no functionally equivalent extraction attacks that are based on the first four types of feedback so far. From the perspective of cryptanalysis, the raw output $f_\theta(x)$ is equivalent to the complete ciphertext corresponding to the plaintext (i.e., the query x). The other four types of feedback only reveal some properties of the ciphertext (raw output $f_\theta(x)$). For example, when $f_\theta(x) \in \mathbb{R}$, the hard-label only tells whether $f_\theta(x) > 0$ holds or not [8]. Among the five types of feedback, the raw output leaks the most information, while the hard-label (i.e., the first type of feedback) leaks the least [11].

Assuming that the feedback of the Oracle is the raw output, Jagielski et al. propose the first functionally equivalent extraction attack against ReLU neural networks with one hidden layer [11], which is extended to deeper neural networks in [18]. At CRYPTO 2020, Carlini et al. propose a differential extraction attack [4] that requires fewer queries than the attack in [18]. However, the differential extraction attack requires an exponential amount of time, which is addressed by Canales-Martínez et al. at EUROCRYPT 2024 [3]. Note that the extraction attacks in [3, 4, 18] are also based on the assumption that the feedback is the raw output. Due to the dependence on the raw output, all the authors in [3, 4, 11, 18], state that the hard-label setting (i.e., the feedback is the first type) is a defense against functionally equivalent extraction.

The above backgrounds lead to the question not studied before

Is it possible to achieve functionally equivalent extraction against neural network models under the hard-label setting?

1.1 Our Results and Techniques

Results. We have addressed this question head-on in this paper. In total, the answer is yes, and we propose the first functionally equivalent extraction attack against ReLU neural networks under the hard-label setting. Here, the definition of functionally equivalent extraction proposed in [4] is extended reasonably.

Definition 1. (Extended Functionally Equivalent Extraction) *The goal of the extended functionally equivalent extraction is to generate a model $f_{\hat{\theta}}$ (i.e., the extracted model), such that $f_{\hat{\theta}}(x) = c \times f_\theta(x)$ holds for all $x \in \mathcal{X}$, where $c > 0$ is a fixed constant, $f_\theta(x)$ and $f_{\hat{\theta}}(x)$ are, respectively, the raw output of the victim model and the extracted model. The extracted model $f_{\hat{\theta}}$ is the functionally equivalent model of the victim model f_θ .*

Since $f_{\hat{\theta}}(x) = c \times f_\theta(x)$ holds for all $x \in \mathcal{X}$, i.e., $f_{\hat{\theta}}$ is a simple scalar product of f_θ , the adversary still can explore the properties of the victim model f_θ by taking $f_{\hat{\theta}}$ as a perfect substitute. This is why we propose this extended definition. From the perspective of cryptanalysis, this extended definition allows the adversary not to guess the 64 bits of the constant c . To evaluate the efficacy of our

model extraction attacks, and quantify the degree to which a model extraction attack has succeeded in practice, we generalize the metric named (ε, δ) -functional equivalence proposed in [4].

Definition 2. (Extended (ε, δ) -Functional Equivalence) *Two models $f_{\hat{\theta}}$ and f_{θ} are (ε, δ) -functional equivalent on \mathcal{S} if there exists a fixed constant $c > 0$ such that*

$$\Pr_{x \in \mathcal{S}} [|f_{\hat{\theta}}(x) - c \times f_{\theta}(x)| \leq \varepsilon] \geq 1 - \delta$$

In this paper, we propose two model extraction attacks, one of which applies to 0-deep neural networks, and the other one applies to k -deep neural networks. The former attack is the basis of the latter attack. Our model extraction attacks theoretically achieve functionally equivalent extraction described in Definition 1, where the constant $c > 0$ is determined by the model parameter θ .

We have also performed numerous experiments on both untrained and trained neural networks, for verifying the effectiveness of our model extraction attacks in practice. The untrained neural networks are obtained by randomly generating model parameters. To fully verify our attacks, we also adopt two real benchmarking image datasets (i.e., MNIST and CIFAR10) widely used in computer vision, and train many classifiers (i.e., trained neural networks) as the victim model. Our model extraction attacks show good performances in experiments. The complete experiment results refer to Tables 1 and 2 in Sect. 7. The number of parameters of neural networks in our experiments is up to 10^5 , but the runtime of the proposed extraction attack on a single core is within several hours. Our experiment code is uploaded to GitHub (https://github.com/AI-Lab-Y/NN_cryptanalytic_extraction).

The analysis of the attack complexity is presented in Appendix B. For the extraction attack on k -deep neural networks, its query complexity is about $\mathcal{O}\left(d_0 \times 2^n \times \log_2 \frac{1}{\varepsilon}\right)$, where d_0 and n are, respectively, the input dimension (i.e., the size of x) and the number of neurons, ε is a precision chosen by the adversary. The computation complexity is about $\mathcal{O}\left(n \times 2^{n^2+n+k}\right)$, where n is the number of neurons and k is the number of hidden layers. The computation complexity of our attack is much lower than that of brute-force searching.

Techniques. By introducing two new concepts, namely model activation pattern and model signature, we obtained some findings as follows. A ReLU neural network is composed of a certain number of affine transformations corresponding to model activation patterns. Each affine transformation leaks partial information about neural network parameters, which is determined by the corresponding model activation pattern. Most importantly, for a neural network that contains n neurons, $n + 1$ special model activation patterns will leak all the information about the neural network parameters.

Inspired by the above findings, we design a series of methods to find decision boundary points, recover the corresponding affine transformations, and further extract the neural network parameters. These methods compose the complete model extraction attacks.

Organization. The basic notations, threat model, attack goal and assumptions are introduced in Sect. 2. Section 3 introduces some auxiliary concepts. Then we introduce the overview of our model extraction attacks, the idealized model extraction attacks, and some refinements in practice in the following three sections respectively. Experiments are introduced in Sect. 7. At last, we present more discussions about our work and conclude this paper.

2 Preliminaries

2.1 Basic Definitions and Notations

This section presents some necessary definitions and notations.

Definition 3. (*k*-Deep Neural Network [4]) A *k*-deep neural network $f_\theta(x)$ is a function parameterized by θ that takes inputs from an input space \mathcal{X} and returns values in an output space \mathcal{Y} . The function $f: \mathcal{X} \rightarrow \mathcal{Y}$ is composed of alternating linear layers f_i and a non-linear activation function σ :

$$f = f_{k+1} \circ \sigma \circ \dots \circ \sigma \circ f_2 \circ \sigma \circ f_1. \quad (1)$$

In this paper, we exclusively study neural networks over $\mathcal{X} = \mathbb{R}^{d_0}$ and $\mathcal{Y} = \mathbb{R}^{d_{k+1}}$, where d_0 and d_{k+1} are positive integers. As in [3,4], we only consider neural networks using the ReLU [15] activation function, given by $\sigma: x \mapsto \max(x, 0)$.

Definition 4. (Fully Connected Layer [4]) The *i*-th fully connected layer of a neural network is a function $f_i: \mathbb{R}^{d_{i-1}} \rightarrow \mathbb{R}^{d_i}$ given by the affine transformation

$$f_i(x) = A^{(i)}x + b^{(i)}. \quad (2)$$

where $A^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ is a $d_i \times d_{i-1}$ weight matrix, $b^{(i)} \in \mathbb{R}^{d_i}$ is a d_i -dimensional bias vector.

Definition 5. (Neuron [4]) A neuron is a function determined by the corresponding weight matrix, bias vector, and activation function. Formally, the *j*-th neuron of layer *i* is the function η given by

$$\eta(x) = \sigma \left(A_j^{(i)}x + b_j^{(i)} \right), \quad (3)$$

where $A_j^{(i)}$ and $b_j^{(i)}$ denote, respectively, the *j*-th row of $A^{(i)}$ and *j*-th coordinate of $b^{(i)}$. In a *k*-deep neural network, there are a total of $\sum_{i=1}^k d_i$ neurons.

Definition 6. (Neuron State [3]) Let $\mathcal{V}(\eta; x)$ denote the value that neuron η takes with $x \in \mathcal{X}$ before applying σ . If $\mathcal{V}(\eta; x) > 0$, then η is active, i.e., the neuron state is active. Otherwise, the neuron state is inactive². The state of the *j*-th neuron in layer *i* on input x is denoted by $\mathcal{P}_j^{(i)}(x) \in \mathbb{F}_2$. If $\mathcal{P}_j^{(i)}(x) = 1$, the neuron is active. If $\mathcal{P}_j^{(i)}(x) = 0$, the neuron is inactive.

² In [3,4], the authors defined one more neuron state, namely critical, i.e., $\mathcal{V}(\eta; x) = 0$, which is a special inactive state since the output of neuron η is 0.

Definition 7. (Neural Network Architecture [4]) *The architecture of a fully connected neural network captures the structure of f_θ : (a) the number of layers, (b) the dimension d_i of each layer $i = 0, \dots, k+1$. We say that d_0 is the dimension of the input to the neural network, and d_{k+1} denotes the number of outputs of the neural network.*

Definition 8. (Neural Network Parameters [4]) *The parameters θ of a k -deep neural network f_θ are the concrete assignments to the weights $A^{(i)}$ and biases $b^{(i)}$ for $i \in \{1, 2, \dots, k+1\}$.*

When neural networks work under the hard-label setting, the raw output $f_\theta(x)$ is processed before being returned [8]. This paper considers the most common processing. The raw output $f_\theta(x) \in \mathbb{R}^{d_{k+1}}$ is first transformed into a category probability vector $\mathbf{P} \in \mathbb{R}^{d_{k+1}}$ by applying the Sigmoid (when $d_{k+1} = 1$) or Softmax (when $d_{k+1} > 1$) function to $f_\theta(x)$ [6]. Then, the category with the largest probability is returned as a hard-label. Definition 9 summarizes the hard-label and corresponding decision conditions on the raw output $f_\theta(x)$.

Definition 9. (Hard-Label) *Consider a k -deep neural network $f: \mathcal{X} \rightarrow \mathcal{Y}$ where $\mathcal{Y} \in \mathbb{R}^{d_{k+1}}$. The hard-label (denoted by z) is related to the outputs $f_\theta(x)$. When $d_{k+1} = 1$, the hard-label $z(f_\theta(x))$ is computed as*

$$z(f_\theta(x)) = \begin{cases} 1, & \text{if } f_\theta(x) > 0, \\ 0, & \text{if } f_\theta(x) \leq 0. \end{cases} \quad (4)$$

When $d_{k+1} > 1$, the output $f_\theta(x)$ is a d_{k+1} -dimensional vector. The hard-label $z(f_\theta(x))$ is the coordinate of the maximum of $f_\theta(x)$.³

2.2 Adversarial Goals and Assumptions

There are two parties in a model extraction attack: the oracle \mathcal{O} who possesses the neural network $f_\theta(x)$, and the adversary who generates queries x to the Oracle. Under the hard-label setting, the Oracle \mathcal{O} returns the hard-label $z(f_\theta(x))$ in Definition 9.

Definition 10. (Model Parameter Extraction Attack) *A model parameter extraction attack receives Oracle access to a parameterized function f_θ (i.e., a k -deep neural network in our paper) and the architecture of f_θ , and returns a set of parameters $\hat{\theta}$ with the goal that $f_{\hat{\theta}}(x)$ is as similar as possible to $c \times f_\theta(x)$ where $c > 0$ is a fixed constant.*

In this paper, we use the $\hat{\cdot}$ symbol to indicate an extracted parameter. For example, θ is the parameters of the victim model f_θ , and $\hat{\theta}$ stands for the parameters of the extracted model $f_{\hat{\theta}}$.

Assumptions. We make the following assumptions of the Oracle \mathcal{O} and the capabilities of the attacker:

³ If there are ties, i.e., multiple items of $f_\theta(x)$ share the same maximum, the hard-label is the smallest one of the coordinates of these items.

- **Architecture knowledge.** We require knowledge of the neural network architecture.
- **Full-domain inputs.** We can feed arbitrary inputs from $\mathcal{X} = \mathbb{R}^{d_0}$.
- **Precise computations.** f_θ is specified and evaluated using 64-bit floating-point arithmetic.
- **Scalar outputs.** The output dimensionality is 1, i.e., $\mathcal{Y} = \mathbb{R}$.⁴
- **ReLU Activations.** All activation functions (σ 's) are the ReLU function.

Compared with the work in [4], we remove the assumption of requiring the raw output $f_\theta(x)$ of the neural network. Now, after querying the Oracle \mathcal{O} , the attacker obtains the hard-label $z(f_\theta(x))$. In other words, the attacker only knows whether $f_\theta(x) > 0$ holds or not.

3 Auxiliary Concepts

To help understand our attacks, this paper proposes some auxiliary concepts.

3.1 Model Activation Pattern

To describe all the neuron states, we introduce a new concept named *Model Activation Pattern*.

Definition 11. (Model Activation Pattern) *Consider a k -deep neural network f_θ with $n = \sum_{i=1}^k d_i$ neurons. The model activation pattern of f_θ over an input $x \in \mathcal{X}$ is a global description of the n neuron states, and is denoted by $\mathcal{P}(x) = (\mathcal{P}^{(1)}(x), \dots, \mathcal{P}^{(k)}(x))$ where $\mathcal{P}^{(i)}(x) \in \mathbb{F}_2^{d_i}$ is the concatenation of d_i neuron states (i.e., $\mathcal{P}_j^{(i)}(x), i \in \{1, \dots, d_i\}$) in layer i .*

In the rest of this paper, the notations $\mathcal{P}_j^{(i)}(x)$, $\mathcal{P}^{(i)}(x)$, and $\mathcal{P}(x)$ are simplified as $\mathcal{P}_j^{(i)}$, $\mathcal{P}^{(i)}$, and \mathcal{P} respectively, when the meaning is clear in context. Besides, $\mathcal{P}^{(i)} \in \mathbb{F}_2^{d_i}$ is represented by a d_i -bit integer. For example, $\mathcal{P}^{(i)} = 2^{j-1}$ means that only the j -th neuron in layer i is active, and $\mathcal{P}^{(i)} = 2^{d_i} - 1$ means that all the d_i neurons are active.

When the model activation pattern is known, one can precisely determine which neural network parameters influence the output $f_\theta(x)$. Consider the j -th neuron η in layer i . Due to the ReLU activation function, if the neuron state is *inactive*, neuron η does not influence the output $f_\theta(x)$. As a result, all the weights $A_{?,j}^{(i+1)}$ and $A_{j,?}^{(i)}$ (i.e., the elements of the j -th column of $A^{(i+1)}$, and the j -th row of $A^{(i)}$ respectively) and the bias $b_j^{(i)}$ do not affect the output $f_\theta(x)$.

⁴ This assumption is fundamental to our work. Our attack only applies to the case of scalar outputs.

Special ‘neuron’. For the convenience of introducing model extraction attacks later, we regard the input $x \in \mathbb{R}^{d_0}$ and the output $f_\theta(x) \in \mathbb{R}$ as, respectively, d_0 and 1 special ‘neurons’ that are always active. So we adopt two extra notations $\mathcal{P}^{(0)} = 2^{d_0} - 1$ and $\mathcal{P}^{(k+1)} = 2^1 - 1 = 1$, for describing the states of the special $d_0 + 1$ ‘neurons’. But if not necessary, we will omit the two notations.

3.2 Model Signature

Consider a k -deep neural network f_θ . For an input $x \in \mathcal{X}$, f_θ can be described as an affine transformation

$$\begin{aligned} f_\theta(x) &= A^{(k+1)} \dots \left(I_{\mathcal{P}}^{(2)} \left(A^{(2)} \left(I_{\mathcal{P}}^{(1)} \left(A^{(1)} x + b^{(1)} \right) \right) + b^{(2)} \right) \right) \dots + b^{(k+1)} \\ &= A^{(k+1)} I_{\mathcal{P}}^{(k)} A^{(k)} \dots I_{\mathcal{P}}^{(2)} A^{(2)} I_{\mathcal{P}}^{(1)} A^{(1)} x + B_{\mathcal{P}} = \Gamma_{\mathcal{P}} x + B_{\mathcal{P}}, \end{aligned} \quad (5)$$

where \mathcal{P} is the model activation pattern over x , $\Gamma_{\mathcal{P}} \in \mathbb{R}^{d_0}$, and $B_{\mathcal{P}} \in \mathbb{R}$. Here, $I_{\mathcal{P}}^{(i)} \in \mathbb{R}^{d_i \times d_i}$ are 0-1 diagonal matrices with a 0 on the diagonal’s j -th entry when the neuron state $\mathcal{P}_j^{(i)}$ is 0, and 1 when $\mathcal{P}_j^{(i)} = 1$.

The affine transformation is denoted by a tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$. Except for \mathcal{P} , the value of the tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ is only determined by the neural network parameters, i.e., $A^{(i)}$ and $b^{(i)}$, $i \in \{1, \dots, k+1\}$. Once the value of any neural network parameters is changed, the value of the tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ corresponding to some \mathcal{P} ’s will change too⁵. Therefore, we regard the set of all the possible tuples $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ as a unique *model signature* of the neural network.

Definition 12. (Model Signature) For a k -deep neural network $f_\theta(x)$, the model signature denoted by \mathcal{S}_θ is the set of affine transformations

$$\mathcal{S}_\theta = \{(\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \text{ for all the } \mathcal{P} \text{'s}\}.$$

In [3], Canales-Martínez et al. use the term ‘signature’ to describe the weights related to a neuron, which is different from the model signature. Except for the model signature, we propose another important concept, namely *normalized model signature*.

Definition 13. (Normalized Model Signature) Consider a victim model f_θ and its model signature $\mathcal{S}_\theta = \{(\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \text{ for all the } \mathcal{P} \text{'s}\}$. Denote by $\Gamma_{\mathcal{P},j}$ the j -th element of $\Gamma_{\mathcal{P}}$ for $j \in \{1, \dots, d_0\}$. Divide the set of \mathcal{P} ’s into two subsets \mathcal{Q}_1 and \mathcal{Q}_2 . For each $\mathcal{P} \in \mathcal{Q}_1$, $\Gamma_{\mathcal{P},j} = 0$ for $j \in \{1, \dots, d_0\}$. For each $\mathcal{P} \in \mathcal{Q}_2$, there is at least one non-zero element in $\Gamma_{\mathcal{P}}$, without loss of generality, assume that $\Gamma_{\mathcal{P},1} \neq 0$. Let $\mathcal{S}_\theta^{\mathcal{N}}$ be the following set

$$\mathcal{S}_\theta^{\mathcal{N}} = \left\{ (\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \text{ for } \mathcal{P} \in \mathcal{Q}_1, \left(\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}, \frac{B_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|} \right) \text{ for } \mathcal{P} \in \mathcal{Q}_2 \right\}.$$

The set $\mathcal{S}_\theta^{\mathcal{N}}$ is the normalized model signature of f_θ .

⁵ We do not consider the case of some neurons being always inactive, since such neurons are redundant and usually deleted by various network pruning methods (e.g., [10]) before the neural network is deployed as a prediction service.

Shortly, the difference between the normalized model signature \mathcal{S}_θ^N and the initial model signature \mathcal{S}_θ is as follows. For each $\mathcal{P} \in \mathcal{Q}_2$, i.e., there is at least one non-zero element in $\Gamma_{\mathcal{P}}$ (without loss of generality, assume that the first element is non-zero, i.e., $\Gamma_{\mathcal{P},1} \neq 0$), we transform the parameter tuple into $\left(\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}, \frac{B_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}\right)$.

In our attacks, the normalized model signature plays two important roles. First, the recovery of all the weights $A^{(i)}$ relies on the subset \mathcal{Q}_2 . Second, our attacks will produce many extracted models during the attack process while at most only one is the functionally equivalent model of f_θ , and the normalized model signature is used to filter functionally inequivalent models.

3.3 Decision Boundary Point

Our attacks exploit a special class of inputs named *Decision Boundary Points*.

Definition 14. (Decision Boundary Point) Consider a neural network f_θ . If an input x makes $f_\theta(x) = 0$ hold, x is a decision boundary point.

The extraction attacks presented at CRYPTO 2020 [4] and EUROCRYPT 2024 [3] exploit a class of inputs, namely critical points. Figure 1 shows the difference between critical points and decision boundary points.

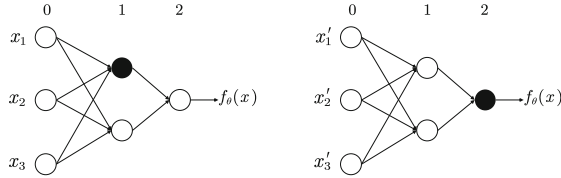


Fig. 1. Left: the critical point $x = [x_1, x_2, x_3]^\top$ makes the output of one neuron (e.g., the solid black circle) 0. Right: the decision boundary point $x' = [x'_1, x'_2, x'_3]^\top$ makes the output of the neural network 0.

Critical points leak information on the neuron states, i.e., whether the output of a neuron is 0, which is the core reason why the differential extraction attack can efficiently extract model parameters [4]. As a comparison, decision boundary points do not leak information on the neuron states.

Finding critical points relies on computing partial derivatives based on the raw output $f_\theta(x)$, refer to the work in [3, 4]. Thus, under the hard-label setting, we can not exploit critical points.

4 Overview of Our Cryptanalytic Extraction Attacks

Under the hard-label setting, i.e., the Oracle returns the most likely class $z(f_\theta(x))$ instead of the raw output $f_\theta(x)$, only decision boundary points x

will leak the value of $f_\theta(x)$, since $f_\theta(x) = 0$. Motivated by this truth, our cryptanalytic extraction attacks focus on decision boundary points.

Attack Process. At a high level, the complete attack contains five steps.

- **Step 1: collect decision boundary points.** The algorithm for finding decision boundary points will be introduced in Sect. 6.1. Suppose that M decision boundary points are collected.
- **Step 2: recover the normalized model signature.** Recover the tuples $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ corresponding to the M decision boundary points. After filtering duplicate tuples, regard the set of the remaining tuples as the (partial) normalized model signature \mathcal{S}_θ^N . Suppose that the size of \mathcal{Q}_2 is N , refer to Definition 13. It means that there are N decision boundary points that can be used to recover weights $A^{(i)}$.
- **Step 3: recover weights layer by layer.** Suppose that there are $n = \sum_{i=1}^k d_i$ neurons in the neural network. Randomly choose $n + 1$ out of N decision boundary points each time, assign a specific model activation pattern \mathcal{P} to each selected decision boundary point, and recover the weights $A^{(1)}, \dots, A^{(k+1)}$.
- **Step 4: recover all the biases.** Based on recovered weights, recover all the biases $b^{(i)}, i \in \{1, \dots, k + 1\}$ simultaneously.
- **Step 5: filter functionally inequivalent models.** As long as $N \geq n + 1$ holds, we will obtain many extracted models, but it is expected that at most only one is the functionally equivalent model. Thus, we filter functionally inequivalent models in this step.

Some functionally inequivalent models may not be filtered. For each surviving extracted model, we test the *Prediction Matching Ratio* (PMR, introduced in Sect. 6.3) over randomly generated inputs, and take the one with the highest PMR as the final candidate.

In Step 2, we recover the tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ by the extraction attack on 0-deep neural networks. In Step 3, for layer $i > 1$, the weight vector $A_j^{(i)}$ of the j -th neuron ($j \in \{1, \dots, d_i\}$) is recovered by solving a system of linear equations. For layer 1, except for selecting d_1 decision boundary points, the recovery of the weights $A^{(1)}$ does not use any extra techniques. In Step 4, all the biases are recovered by solving a system of linear equations.

5 Idealized Hard-Label Model Extraction Attack

This section introduces $(0, 0)$ -functionally equivalent model extraction attacks under the hard-label setting, which assumes infinite precision arithmetic and recovers the functionally equivalent model. We first introduce the 0-deep neural network extraction attack, which is used in the k -deep neural network extraction attack to recover the normalized model signature.

Note that this section only (partially) involves Steps 2, 3, and 4 introduced in Sect. 4. In the next section, we introduce the remaining steps and refine the idealized attacks to work with finite precision.

5.1 Zero-Deep Neural Network Extraction

According to Definition 3, zero-deep neural networks are affine functions $f_\theta(x) \equiv A^{(1)} \cdot x + b^{(1)}$ where $A^{(1)} \in \mathbb{R}^{d_0}$, and $b^{(1)} \in \mathbb{R}$. Let $A^{(1)} = [w_1^{(1)}, \dots, w_{d_0}^{(1)}]$, and $x = [x_1, x_2, \dots, x_{d_0}]^\top$. The model signature is $\mathcal{S}_\theta = (A^{(1)}, b^{(1)})$.

Our extraction attack is based on a decision boundary point x (i.e., $f_\theta(x) = 0$), and composed of 3 steps: (1) recover weight signs, i.e., the sign of $w_i^{(1)}$; (2) recover weights $w_i^{(1)}$; (3) recover bias $b^{(1)}$.

Recover Weight Signs. Denote by $e_i \in \mathbb{R}^{d_0}$ the basis vector where only the i -th element is 1 and other elements are 0.

Let the decision boundary point x move along the direction $e_i, i \in \{1, \dots, d_0\}$, and the moving stride is $s \in \mathbb{R}$ where $|s| > 0$. Query the Oracle and obtain the hard-label $z(f_\theta(x + se_i))$, then the sign of $w_i^{(1)}$ is

$$\text{sign}(w_i^{(1)}) = \begin{cases} 1, & \text{if } s > 0 \text{ and } z(f_\theta(x + se_i)) = 1, \\ -1, & \text{if } s < 0 \text{ and } z(f_\theta(x + se_i)) = 1. \end{cases} \quad (6)$$

When $z(f_\theta(x + se_i)) = 1$, we have $f_\theta(x + se_i) > 0$, i.e., $w_i^{(1)} \times s > 0$. Thus, the sign of $w_i^{(1)}$ is the same as that of s . If $z(f_\theta(x + se_i)) = 0$ always holds, no matter if s is positive or negative, then we have $w_i^{(1)} = 0$.

Recover Weights. Without loss of generality, assume that $w_1^{(1)} \neq 0$.

At first, let the decision boundary point x move along e_1 with a moving stride s_1 , such that the hard-label of the new point $x + s_1 e_1$ is 1, i.e., $z(f_\theta(x + s_1 e_1)) = 1$. Then, let the new point $x + s_1 e_1$ move along e_i with a moving stride s_i where $i \neq 1$ and $w_i^{(1)} \neq 0$, such that $x + s_1 e_1 + s_i e_i$ is a decision boundary point too. As a result, we have

$$s_1 w_1^{(1)} + s_i w_i^{(1)} = 0, \quad (7)$$

and obtain the weight ratio $\frac{w_i^{(1)}}{w_1^{(1)}}$. Since the signs of $w_i^{(1)}$ are known, the final extracted weights are

$$\widehat{A}^{(1)} = \left[\frac{w_1^{(1)}}{|w_1^{(1)}|}, \frac{w_2^{(1)}}{|w_1^{(1)}|}, \dots, \frac{w_{d_0}^{(1)}}{|w_1^{(1)}|} \right]. \quad (8)$$

Recover Bias. The extracted bias is $\widehat{b}^{(1)} = -\widehat{A}^{(1)} \cdot x = \frac{b^{(1)}}{|w_1^{(1)}|}$.

Thus, the model signature of $f_{\widehat{\theta}}$ is $\mathcal{S}_{\widehat{\theta}} = \left(\frac{A^{(1)}}{|w_1^{(1)}|}, \frac{b^{(1)}}{|w_1^{(1)}|} \right)$, and $f_{\widehat{\theta}}(x) = \frac{f(x)}{|w_1^{(1)}|}$.

Remark 1. In [14], the authors propose different methods to extract the parameters of linear functions $f_\theta(x) = A^{(1)} \cdot x$. Since this paper mainly focuses on the extraction attack on k -deep neural networks, we do not deeply compare our attack with the methods in [14].

5.2 k -Deep Neural Network Extraction

Basing the 0-deep neural network extraction attack, we develop an extraction attack on k -deep neural networks. Recall that, the expression of k -deep neural networks is

$$\begin{aligned} f_{\theta}(x) &= A^{(k+1)} \dots \left(I_{\mathcal{P}}^{(2)} \left(A^{(2)} \left(I_{\mathcal{P}}^{(1)} \left(A^{(1)} x + b^{(1)} \right) \right) + b^{(2)} \right) \right) \dots + b^{(k+1)} \\ &= \Gamma_{\mathcal{P}} x + B_{\mathcal{P}} \end{aligned} \quad (9)$$

where the model activation pattern is $\mathcal{P} = (\mathcal{P}^{(0)}, \mathcal{P}^{(1)}, \dots, \mathcal{P}^{(k)}, \mathcal{P}^{(k+1)})$ and

$$\Gamma_{\mathcal{P}} = A^{(k+1)} I_{\mathcal{P}}^{(k)} A^{(k)} \dots I_{\mathcal{P}}^{(2)} A^{(2)} I_{\mathcal{P}}^{(1)} A^{(1)}. \quad (10)$$

Notations. Our attack recovers weights layer by layer. Assuming that the weights of the first $i - 1$ layers have been recovered and we are trying to recover $A^{(i)}$ where $i \in \{1, \dots, k + 1\}$, we describe k -deep neural networks as:

$$f_{\theta}(x) = \Gamma_{\mathcal{P}} x + B_{\mathcal{P}} = \mathcal{G}^{(i)} A^{(i)} C^{(i-1)} x + B_{\mathcal{P}}, \quad (11)$$

where $\mathcal{G}^{(i)} \in \mathbb{R}^{d_i}$ and $C^{(i-1)} \in \mathbb{R}^{d_{i-1} \times d_0}$ are, respectively, related to the unrecovered part (excluding $A^{(i)}$) and recovered part of the neural network f_{θ} .

The values of $\mathcal{G}^{(i)}$ and $C^{(i-1)}$ are

$$\begin{aligned} \mathcal{G}^{(i)} &= \begin{cases} A^{(k+1)} I_{\mathcal{P}}^{(k)} A^{(k)} \dots I_{\mathcal{P}}^{(i+1)} A^{(i+1)} I_{\mathcal{P}}^{(i)}, & \text{if } i \in \{1, \dots, k\} \\ 1, & \text{if } i = k + 1 \end{cases} \\ C^{(i-1)} &= \begin{cases} I, & \text{if } i = 1 \\ I_{\mathcal{P}}^{(i-1)} A^{(i-1)} \dots I_{\mathcal{P}}^{(1)} A^{(1)}, & \text{if } i \in \{2, \dots, k + 1\} \end{cases} \end{aligned} \quad (12)$$

where $C^{(0)} = I \in \mathbb{R}^{d_0 \times d_0}$ is a diagonal matrix with a 1 on each diagonal entry.

Core Idea of Recovering Weights Layer by Layer. To better grasp the attack details presented later, we first introduce the core idea of recovering weights layer by layer. Assuming that the extracted weights of the first $i - 1$ layers are known, i.e., $\widehat{A}^{(1)}, \dots, \widehat{A}^{(i-1)}$ are known, we try to recover the weights in layer i .

To obtain the weight vector $\widehat{A}_j^{(i)}$ of the j -th neuron (denoted by $\eta_j^{(i)}$) in layer $i \in \{1, \dots, k + 1\}$ ⁶, we exploit a decision boundary point with the model activation pattern $\mathcal{P} = (\mathcal{P}^{(0)}, \mathcal{P}^{(1)}, \dots, \mathcal{P}^{(k)}, \mathcal{P}^{(k+1)})$ where

$$\mathcal{P}^{(i-1)} = 2^{d_{i-1}} - 1, \quad \mathcal{P}^{(i)} = 2^{j-1}. \quad (13)$$

It means that, in layer i , only the j -th neuron is active, and all the d_{i-1} neurons in layer $i - 1$ are active. Figure 2 shows a schematic diagram under this scenario.

Since $\mathcal{P}^{(i)} = 2^{j-1}$, all the $k - i$ layers starting from layer $i + 1$ collapse into a direct connection from $\eta_j^{(i)}$ to the output $f_{\theta}(x)$. The weight of this connection is

⁶ When $i = k + 1$, it means that we are trying to recover the weights $A^{(k+1)}$.

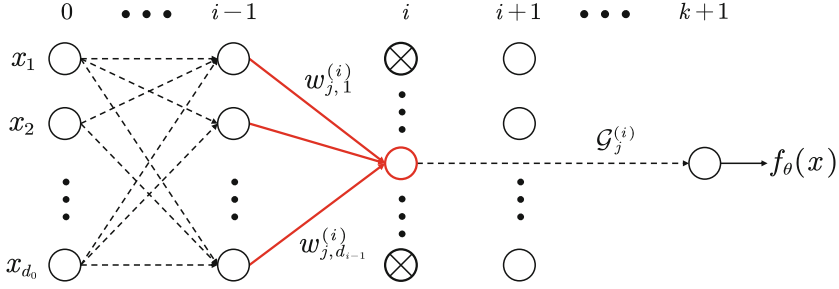


Fig. 2. The core idea of recovering the weight vector of the j -th neuron in layer i . Let $x = [x_1, \dots, x_{d_0}]^\top$ be a decision boundary point with $\mathcal{P}^{(i-1)} = 2^{d_{i-1}} - 1$, $\mathcal{P}^{(i)} = 2^{d_i} - 1$, i.e., in layer i , only the j -th neuron (the red hollow circle) is active, and in layer $i - 1$, all the neurons are active. The first $i - 1$ layers have been extracted, and collapse into one layer. All the layers starting from layer $i + 1$ collapse into a direct connection between the j -th neuron in layer i and the final output.

$\mathcal{G}_j^{(i)}$, i.e., the j -th element of $\mathcal{G}^{(i)}$ (see Eq. (12)). The expression (see Eq. (11)) of the k -deep neural network further becomes

$$f_\theta(x) = \Gamma_{\mathcal{P}} \cdot x + B_{\mathcal{P}} = \mathcal{G}_j^{(i)} A_j^{(i)} \cdot C^{(i-1)} \cdot x + B_{\mathcal{P}},$$

where $\mathcal{G}_j^{(i)} \in \mathbb{R}$ and $A_j^{(i)} \cdot C^{(i-1)} \in \mathbb{R}^{d_0}$.

In Step 2 (see Sect. 4), applying the extraction attack on zero-deep neural networks, we can obtain the tuple $\left(\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}, \frac{B_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|} \right)$ where

$$\Gamma_{\mathcal{P}} = \mathcal{G}_j^{(i)} A_j^{(i)} \cdot C^{(i-1)}, \quad \Gamma_{\mathcal{P},v} = \mathcal{G}_j^{(i)} A_j^{(i)} \cdot C_{\cdot,v}^{(i-1)}. \tag{14}$$

Here the symbol $C_{\cdot,v}^{(i-1)}$ stands for the v -th column vector of $C^{(i-1)}$.

According to Eq. (14), the value of each element of $\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}$ is not related to the absolute value of $\mathcal{G}_j^{(i)}$, i.e., the unrecovered part does not affect the affine transformation. Then, basing the vector $\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}$ and the extracted weights $\hat{A}^{(1)}, \dots, \hat{A}^{(i-1)}$, we build a system of linear equations and solve it to obtain $\hat{A}_j^{(i)}$. Next, we introduce more attack details.

Recover Weights in Layer 1. To recover the weight vector of the j -th neuron in layer 1, we exploit the model activation pattern \mathcal{P} where

$$\mathcal{P}^{(1)} = 2^{j-1}; \quad \mathcal{P}^{(i)} = 2^{d_i} - 1, \text{ for } i \in \{0, 2, 3, \dots, k + 1\}. \tag{15}$$

It means that, in layer 1, only the j -th neuron is active, and all the neurons in other layers are active.

Under this model activation pattern, according to Eq. (12), we have

$$\mathcal{G}^{(1)} = A^{(k+1)} A^{(k)} \dots A^{(2)} \Gamma_{\mathcal{P}}^{(1)}. \tag{16}$$

Now, the expression of the k -deep neural network is

$$f_{\theta}(x) = \mathcal{G}_j^{(1)} \left(A_j^{(1)} x + b_j^{(1)} \right) + B_{(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})} = \mathcal{G}_j^{(1)} A_j^{(1)} x + B_{\mathcal{P}} \quad (17)$$

where $A_j^{(1)} = [w_{j,1}^{(1)}, \dots, w_{j,d_0}^{(1)}]$, $\mathcal{G}_j^{(1)} \in \mathbb{R}$ is the j -th element of $\mathcal{G}^{(1)}$. As for $B_{(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})} \in \mathbb{R}$, it is a constant determined by $(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})$. In other words, when $\mathcal{P}^{(1)}$ changes, the value of $B_{(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})}$ does not change.

Recall that, in Step 2, we have recovered the following weight vector

$$\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|} = \left[\frac{\mathcal{G}_j^{(1)} w_{j,1}^{(1)}}{|\mathcal{G}_j^{(1)} w_{j,1}^{(1)}|}, \dots, \frac{\mathcal{G}_j^{(1)} w_{j,d_0}^{(1)}}{|\mathcal{G}_j^{(1)} w_{j,1}^{(1)}|} \right], j \in \{1, \dots, d_1\}. \quad (18)$$

In this step, our target is to obtain $\widehat{A}_j^{(1)}$ where

$$\widehat{A}_j^{(1)} = [\widehat{w}_{j,1}^1, \dots, \widehat{w}_{j,d_0}^1] = \left[\frac{w_{j,1}^1}{|w_{j,1}^1|}, \dots, \frac{w_{j,d_0}^1}{|w_{j,1}^1|} \right], j \in \{1, \dots, d_1\}. \quad (19)$$

Therefore, we need to determine d_1 signs, i.e., the signs of $\mathcal{G}_j^{(1)}$ for $\mathcal{P}^{(1)} = 2^{j-1}$ where $j \in \{1, \dots, d_1\}$.

Since $A_j^{(1)} x + b_j^{(1)} > 0$, we know that $\mathcal{G}_j^{(1)} \times B_{(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})} < 0$ holds for $j \in \{1, \dots, d_1\}$, which tells us that the above d_1 signs are the *same*. Thus, by guessing 1 sign, i.e., the sign of $\mathcal{G}_j^{(1)}$ for $\mathcal{P}^{(1)} \in \{2^{1-1}, \dots, 2^{d_1-1}\}$, we obtain d_1 weight vectors presented in Eq. (19).

Recover Weights in Layer i ($i > 1$). To recover the weight vector of the j -th neuron in layer i , we exploit the model activation pattern \mathcal{P} where

$$\mathcal{P}^{(i)} = 2^{j-1}; \mathcal{P}^{(q)} = 2^{d_q} - 1, \text{ for } q \in \{0, \dots, i-1, i+1, \dots, k+1\}. \quad (20)$$

It means that, in layer i , only the j -th neuron is active, and all the neurons in other layers are active.

Under this model activation pattern, according to Eq. (12), we have

$$\begin{aligned} \mathcal{G}^{(i)} &= \begin{cases} A^{(k+1)} A^{(k)} \dots A^{(i+2)} A^{(i+1)} I_{\mathcal{P}}^{(i)}, & \text{if } i \in \{2, \dots, k\}, \\ 1, & \text{if } i = k+1, \end{cases} \\ C^{(i-1)} &= A^{(i-1)} A^{(i-2)} \dots A^{(1)}, \text{ if } i \in \{2, \dots, k+1\}. \end{aligned} \quad (21)$$

Now, the expression of k -deep neural networks becomes

$$\begin{aligned} f_{\theta}(x) &= \mathcal{G}_j^{(i)} \left(A_j^{(i)} C^{(i-1)} x + B_{(\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(i)})} \right) + B_{(\mathcal{P}^{(i+1)}, \dots, \mathcal{P}^{(k)})} \\ &= \mathcal{G}_j^{(i)} A_j^{(i)} C^{(i-1)} x + B_{\mathcal{P}}, \end{aligned} \quad (22)$$

where $A_j^{(i)} = [w_{j,1}^{(i)}, \dots, w_{j,d_{i-1}}^{(i)}]$, $\mathcal{G}_j^{(i)} \in \mathbb{R}$ and $C^{(i-1)} \in \mathbb{R}^{d_{i-1} \times d_0}$. Besides, $B_{(\mathcal{P}^{(i+1)}, \dots, \mathcal{P}^{(k)})} \in \mathbb{R}$ is not related to $(\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(i)})$, and only determined by $(\mathcal{P}^{(i+1)}, \dots, \mathcal{P}^{(k)})$, i.e., $B_{(\mathcal{P}^{(i+1)}, \dots, \mathcal{P}^{(k)})}$ is the same constant for $j \in \{1, \dots, d_i\}$.

Let us further rewrite $f_\theta(x)$ in Eq. (22) as

$$f_\theta(x) = \mathcal{G}_j^{(i)} \left(\left(\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right) x_1 + \dots + \left(\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,d_0}^{(i-1)} \right) x_{d_0} \right) + B_{\mathcal{P}} \quad (23)$$

where $C_{v,u}^{(i-1)} \in \mathbb{R}$ is the u -th element of the v -th row vector of $C^{(i-1)}$.

In Step 2, using the zero-deep neural network extraction attack, we have recovered the following d_i weight vectors ($j \in \{1, \dots, d_i\}$)

$$\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|} = \left[\frac{\mathcal{G}_j^{(i)} \left(\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right)}{\left| \mathcal{G}_j^{(i)} \left(\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right) \right|}, \dots, \frac{\mathcal{G}_j^{(i)} \left(\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,d_0}^{(i-1)} \right)}{\left| \mathcal{G}_j^{(i)} \left(\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right) \right|} \right]. \quad (24)$$

In this step, our target is to obtain the weight vector $\hat{A}_j^{(i)} = [\hat{w}_{j,1}^{(i)}, \dots, \hat{w}_{j,d_{i-1}}^{(i)}]$.

It is clear that we need to guess the sign of $\mathcal{G}_j^{(i)}$ for $j \in \{1, \dots, d_i\}$. Again, all the d_i signs are the same. Consider the expression in Eq. (22). Since the j -th neuron is active, its output exceeds 0, i.e.,

$$A_j^{(i)} C^{(i-1)} x + B_{(\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(i)})} > 0, j \in \{1, \dots, d_i\}.$$

Then $\mathcal{G}_j^{(i)} \times B_{(\mathcal{P}^{(i+1)}, \dots, \mathcal{P}^{(k)})} < 0$ holds for $j \in \{1, \dots, d_i\}$. At the same time, since $B_{(\mathcal{P}^{(i+1)}, \dots, \mathcal{P}^{(k)})}$ is a constant, all the d_i signs are the same. Therefore, by guessing one sign, i.e., the sign of $\mathcal{G}_j^{(i)}$, based on Eq. (24), we obtain

$$\left[\frac{\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)}}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \dots, \frac{\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,d_0}^{(i-1)}}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|} \right], j \in \{1, \dots, d_i\}. \quad (25)$$

Note that $\hat{C}_{v,u}^{(i-1)}$ can be obtained using Eq. (21), since $\hat{A}^{(1)}, \dots, \hat{A}^{(i-1)}$ are known. Then, basing the vector in Eq. (25), we build a system of linear equations

$$\begin{cases} \sum_{v=1}^{d_{i-1}} \hat{w}_{j,v}^{(i)} \hat{C}_{v,1}^{(i-1)} = \frac{\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)}}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \\ \vdots \\ \sum_{v=1}^{d_{i-1}} \hat{w}_{j,v}^{(i)} \hat{C}_{v,d_0}^{(i-1)} = \frac{\sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,d_0}^{(i-1)}}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \end{cases} \quad (26)$$

When $d_0 \geq d_{i-1}$ ⁷, we obtain $\widehat{A}_j^{(i)} = [\widehat{w}_{j,1}^{(i)}, \dots, \widehat{w}_{j,d_{i-1}}^{(i)}]$ by solving the above system of linear equations. Lemma 1 summarizes the expression of extracted weight vectors $\widehat{A}_j^{(i)}, j \in \{1, \dots, d_i\}, i \in \{2, \dots, k+1\}$.

Lemma 1. *Based on the system of linear equations presented in Eq. (26), for $i \in \{2, \dots, k+1\}$ and $j \in \{1, \dots, d_i\}$, the extracted weight vector $\widehat{A}_j^{(i)} = [\widehat{w}_{j,1}^{(i)}, \dots, \widehat{w}_{j,d_{i-1}}^{(i)}]$ is*

$$\widehat{A}_j^{(i)} = \left[\frac{w_{j,1}^{(i)} \times \left| \sum_{v=1}^{d_{i-2}} w_{1,v}^{(i-1)} C_{v,1}^{(i-2)} \right|}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \dots, \frac{w_{j,d_{i-1}}^{(i)} \times \left| \sum_{v=1}^{d_{i-2}} w_{d_{i-1},v}^{(i-1)} C_{v,1}^{(i-2)} \right|}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|} \right], \quad (27)$$

where $C_{v,1}^{(q)} = A_v^{(q)} A^{(q-1)} \dots A^{(2)} [A_{1,1}^{(1)}, \dots, A_{d_1,1}^{(1)}]^\top$.

Proof. The proof refers to Appendix A.

In Lemma 1, for the consistency of the mathematical symbols, the weights $A^{(k+1)}$ are denoted by $[w_{1,1}^{(k+1)}, \dots, w_{1,d_k}^{(k+1)}]$ instead of $[w_1^{(k+1)}, \dots, w_{d_k}^{(k+1)}]$.

Recover All the Biases. Since $\widehat{A}^{(i)}$ for $i \in \{1, \dots, k+1\}$ have been obtained, we can extract all the biases by solving a system of linear equations.

Concretely, for the $\sum_{i=1}^k d_i + 1$ decision boundary points, $f_{\widehat{\theta}}(x) = 0$ should hold. Thus, we build a system of linear equations: $f_{\widehat{\theta}}(x) = 0$ where the expression of $f_{\widehat{\theta}}(x)$ refers to Eq. (9). Combining with Lemma 1, by solving the above system, we will obtain

$$\begin{cases} \widehat{b}^{(i)} = \left[\frac{b_1^{(i)}}{\left| \sum_{v=1}^{d_{i-1}} w_{1,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \dots, \frac{b_{d_i}^{(i)}}{\left| \sum_{v=1}^{d_{i-1}} w_{d_i,v}^{(i)} C_{v,1}^{(i-1)} \right|} \right], i \in \{1, \dots, k\} \\ \widehat{b}^{(k+1)} = \frac{b^{(k+1)}}{\left| \sum_{v=1}^{d_k} w_v^{(k+1)} C_{v,1}^{(k)} \right|}. \end{cases} \quad (28)$$

Based on the extracted neural network parameters (see Eq. (19), Eq. (27), and Eq. (28)), the model signature of the extracted model $f_{\widehat{\theta}}$ is

$$\mathcal{S}_{\widehat{\theta}} = \left\{ (\widehat{\Gamma}_{\mathcal{P}}, \widehat{B}_{\mathcal{P}}) = \left(\frac{\Gamma_{\mathcal{P}}}{\left| \sum_{v=1}^{d_k} w_v^{(k+1)} C_{v,1}^{(k)} \right|}, \frac{B_{\mathcal{P}}}{\left| \sum_{v=1}^{d_k} w_v^{(k+1)} C_{v,1}^{(k)} \right|} \right) \text{ for all the } \mathcal{P}'\text{s} \right\}.$$

Consider the j -th neuron η in layer i . For an input $x \in \mathcal{X}$, denote by $h(\eta; x)$ the output of the neuron of the victim model. Based on the extracted neural network parameters (see Eq. (19), Eq. (27), and Eq. (28)), the output of the neuron of the extracted model is $\frac{h(\eta; x)}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}$. At the same time, all the

⁷ The case of $d_0 \geq d_{i-1}$ is common in various applications, particularly in computer vision [9, 13, 17], since the dimensions of images or videos are often large.

weights $w_{\cdot,j}^{(i+1)}$ in layer $i + 1$ are increased by a factor of $\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|$. Thus, for the victim model and extracted model, the j -th neuron in layer i has the same influence on all the neurons in layer $i + 1$. As a result, for any $x \in \mathcal{X}$, the model activation pattern of the victim model is the same as that of the extracted model. Combining with $\mathcal{S}_{\hat{\theta}}$, for all $x \in \mathcal{X}$, we have

$$f_{\hat{\theta}}(x) = \frac{1}{\left| \sum_{v=1}^{d_k} w_v^{(k+1)} C_{v,1}^{(k)} \right|} \times f_{\theta}(x). \quad (29)$$

In Appendix C, we apply the extraction attack on 1-deep neural networks and directly present the extracted model, which helps further understand our attack.

Remark 2. Except for the $n + 1$ model activation patterns as shown in Eq. (15) and Eq. (20), the adversary could choose a new set of $n + 1$ model activation patterns. The reason is as follows. Consider the recovery of the weight vector of the j -th neuron in layer i , and look at Fig. 2 again. Our attack only requires that: (1) in layer i , only the j -th neuron is active; (2) in layer $i - 1$, all the d_{i-1} neurons are active. The neuron states in other layers do not affect the attack. Thus, there are more options for the $n + 1$ model activation patterns, and the rationale does not change.

Discussion on The Computation Complexity. Once $n + 1$ decision boundary points and k sign guesses are selected, to obtain an extracted model, we just need to solve $n + 2 - d_1$ systems of linear equations. However, since the model activation pattern of a decision boundary point is unknown, we have to traverse all the possible combinations of $n + 1$ decision boundary points (see Step 3 in Sect. 4), which is the bottleneck of the total computation complexity. The complete analysis of the attack complexity is presented in Appendix B.

6 Instantiating the Extraction Attack in Practice

Recall that, the complete extraction attack contains 5 steps introduced in Sect. 4. To obtain a functionally equivalent model, the adversary also needs three auxiliary techniques: *finding decision boundary points* (related to Steps 1 and 2), *filtering duplicate affine transformations* (related to Step 2), and *filtering functionally inequivalent models* (related to Step 5).

The idealized extraction attack introduced in Sect. 5 relies on decision boundary points x that make $f_{\theta}(x) = 0$ strictly hold. This section will propose a binary searching method to find decision boundary points under the hard-label setting. Under finite precision, it is hard to find decision boundary points x that make $f_{\theta}(x) = 0$ strictly hold. Therefore, the proposed method returns input points x close to the decision hyperplane as decision boundary points. As a result, the remaining two techniques need to consider the influence of finite precision. This ensures our model extraction attacks work in practice, for producing a $(\varepsilon, 0)$ -functionally equivalent model.

6.1 Finding Decision Boundary Points

Let us see how to find decision boundary points under the hard-label setting. Figure 3 shows a schematic diagram in a 2-dimensional input space.

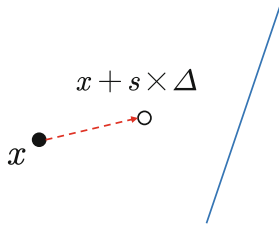


Fig. 3. A schematic diagram of finding decision boundary points. The blue solid line stands for the decision hyperplane composed of decision boundary points. The red dashed line stands for a direction vector $\Delta \in \mathbb{R}^{d_0}$. The starting point $x \in \mathbb{R}^{d_0}$ (i.e., the solid black circle) moves along the direction Δ , and arrives at $x + s \times \Delta$ (i.e., the hollow black circle) where $s \in \mathbb{R}$ is the moving stride.

We first randomly pick a starting point $x \in \mathbb{R}^{d_0}$ and non-zero direction vector $\Delta \in \mathbb{R}^{d_0}$. Then let the starting point move along the direction Δ or the opposite direction $-\Delta$. It is expected that the starting point will eventually cross the decision hyperplane in one direction, as long as Δ and $-\Delta$ are not parallel to the decision hyperplane.

Denote by $s \in \mathbb{R}$ the moving stride of the starting point, which means that the starting point arrives at $x + s \times \Delta$. After querying the Oracle with x and $x + s \times \Delta$, if $z(f_\theta(x)) \neq z(f_\theta(x + s \times \Delta))$ (i.e., the two labels are different), we know that the starting point has crossed the decision hyperplane when the moving stride is s . Now, the core of finding decision boundary points is to determine a suitable moving stride s , such that the starting point reaches the decision hyperplane, i.e., $f_\theta(x + s \times \Delta) = 0$ holds.

This task is done by *binary search*. Concretely, randomly choose two different moving strides s_{slow} and s_{fast} at first, such that

$$\begin{aligned} z(f_\theta(x + s_{\text{slow}} \times \Delta)) &= z(f_\theta(x)), \\ z(f_\theta(x + s_{\text{slow}} \times \Delta)) &\neq z(f_\theta(x + s_{\text{fast}} \times \Delta)). \end{aligned} \quad (30)$$

Then, without changing the conditions presented in Eq. (30), we dynamically adjust s_{slow} and s_{fast} until their absolute difference is close to 0, i.e., $|s_{\text{slow}} - s_{\text{fast}}| < \epsilon$ where ϵ is a precision defined by the adversary. Finally, return $x + s_{\text{slow}} \times \Delta$ as a decision boundary point.

Since the precision ϵ is finite, $x + s_{\text{slow}} \times \Delta$ is not strictly at the decision boundary, which will inevitably introduce minor errors (equivalent to noises) into the extracted model. If ϵ decreases, then $x + s_{\text{slow}} \times \Delta$ will be closer to the decision boundary, which is helpful to the model extraction attack, refers to the experiment results in Sect. 7.

6.2 Filtering Duplicate Affine Transformations

For a k -deep neural network f_θ consisting of $n = \sum_{i=1}^k d_i$ neurons, the idealized extraction attack exploits special $n + 1$ model activation patterns.

To ensure that the required $n + 1$ model activation patterns occur with a probability as high as possible, in Step 1 introduced in Sect. 4, we collect M decision boundary points where $M \gg n + 1$, e.g., $M = c_n 2^n$ and c_n is a small factor. As a result, there are many collected decision boundary points with duplicate model activation patterns. Therefore, in Step 2, after recovering the parameter tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ (i.e., the affine transformation) corresponding to each decision boundary point, we need to filter the decision boundary points with duplicate affine transformations, since their model activation patterns should be the same. When filtering duplicate affine transformations, we consider two possible cases.

Filtering Correctly Recovered Affine Transformations. In the first case, assume that two affine transformations are both correctly recovered.

However, recovering affine transformations (i.e., 0-deep neural network extraction attack) relies on finding decision boundary points, which introduces minor errors. This is equivalent to adding noises to the recovered affine transformations, i.e., the tuples $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$. To check whether two noisy affine transformations are the same, we adopt the checking rule below.

Comparing two vectors. Consider two vectors with the same dimension, e.g., $V^1 \in \mathbb{R}^d, V^2 \in \mathbb{R}^d$. Set a small threshold φ . If the following d inequations hold simultaneously

$$|V_j^1 - V_j^2| < \varphi, j \in \{1, \dots, d\} \tag{31}$$

where V_j^1 and V_j^2 are, respectively, the j -th element of V^1 and V^2 , the two vectors are considered to be the same.

Filtering Wrongly Recovered Affine Transformations. In the second case, assume that one affine transformation is correctly recovered and another one is partially recovered.

For the extraction attack on k -deep neural networks, when recovering the affine transformation corresponding to an input by the 0-deep neural network extraction attack (see Sect. 5.1), the process of binary search should not change the model activation pattern. Otherwise, the affine transformation may be wrongly recovered. Recall that, in the 0-deep neural network extraction attack, the d_0 elements of $\Gamma_{\mathcal{P}}$ are recovered one by one independently. Thus, the wrong recovery of one element of $\Gamma_{\mathcal{P}}$ does not influence the recovery of other elements.

As a result, we have to consider the case that one transformation is partially recovered. In this case, the filtering method is as follows. Consider two vectors $V^1 \in \mathbb{R}^d$ and $V^2 \in \mathbb{R}^d$. If $|V_j^1 - V_j^2| < \varphi$ holds for at least $(d - d_\varphi)$ j 's where $j \in \{1, \dots, d\}$ and $d_\varphi \in \mathbb{N}$ is a threshold, the two vectors are considered to be the same. Suppose that the occurrence frequencies of V^1 and V^2 are o_1 and o_2 respectively, we regard V^1 as the correctly recovered affine transformation if $o_1 \gg o_2$, and vice versa.

6.3 Filtering Functionally Inequivalent Extracted Models

Consider k -deep neural networks consisting of $n = \sum_{i=1}^k d_i$ neurons. As introduced in Sect. 4, each time we randomly choose $n + 1$ out of N collected decision boundary points to generate an extracted model. Moreover, according to Sect. 5.2, in the extraction attack, we need to guess k signs, i.e., the sign of $\mathcal{G}_j^{(i)}, i \in \{1, \dots, k\}$.

When the model activation patterns of the selected $n + 1$ decision boundary points are not those required in the extraction attack, or at least one of the k sign guesses is wrong, the resulting extracted model $f_{\hat{\theta}}$ is not a functionally equivalent model of the victim model f_{θ} . Thus, we will get many functionally inequivalent extracted models.

Besides, due to the minor errors introduced by the finite precision used in finding decision boundary points, the parameters of the extracted model may be slightly different from the theoretical values (see Eq. (19), Eq. (27), and Eq. (28)). This subsection introduces three methods to filter functionally inequivalent extracted models, one of which considers the negative influence of finite precision together.

Filtering by the Normalized Model Signature. Before introducing the filtering method, we discuss how many possible model activation patterns there are at most for a k -deep neural network. Lemma 2 answers this question.

Lemma 2. *For a k -deep neural network consisting of $n = \sum_{i=1}^k d_i$ neurons, the upper bound of the number of possible model activation patterns is*

$$H = \left(\prod_{i=1}^k (2^{d_i} - 1) \right) + \sum_{i=2}^k \left(\prod_{j=1}^{i-1} (2^{d_j} - 1) \right), \quad (32)$$

where d_i is the number of neurons in layer i .

Proof. If all the d_i neurons in layer i are inactive, i.e., the outputs of these neurons are 0, then the neuron states of all the $\sum_{j=i+1}^k d_j$ neurons in the last $k - i$ layers are deterministic. In this case, the number of possible model activation patterns is decided by the first $i - 1$ layers, i.e., the maximum is $\prod_{j=1}^{i-1} (2^{d_j} - 1)$. If there is at least one active neuron in each layer, then there are at most $\prod_{i=1}^k (2^{d_i} - 1)$ possible model activation patterns.

After all the weights $\hat{A}^{(i)}$ and biases $\hat{b}^{(i)}, i \in \{1, \dots, k + 1\}$ are obtained, we assume that all the H model activation patterns are possible, and compute the resulting normalized model signature $\mathcal{S}_{\hat{\theta}}^{\mathcal{N}}$. Denote by $\mathcal{S}_{\theta}^{\mathcal{N}}$ the normalized model signature recovered in Step 2 (see Sect. 4). If $\mathcal{S}_{\hat{\theta}}^{\mathcal{N}}$ is not a subset of $\mathcal{S}_{\theta}^{\mathcal{N}}$, we regard $f_{\hat{\theta}}$ as a functionally inequivalent model.

Due to the minor errors caused by finite precision, i.e., the slight difference between the extracted parameters $\hat{\theta}$ and the theoretical values (see Eq. (19), Eq. (27), and Eq. (28)), when checking whether a tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \in \mathcal{S}_{\hat{\theta}}^{\mathcal{N}}$ is equal

to a tuple $(\widehat{I}_{\mathcal{P}}, \widehat{B}_{\mathcal{P}}) \in \mathcal{S}_{\theta}^{\mathcal{N}}$ or not, we adopt the checking rule presented in Sect. 6.2, refers to Eq. (31).

Besides, the filtering method in Sect. 6.2 does not ensure that all the wrongly recovered affine transformations are filtered. To avoid the functionally equivalent model being filtered, we adopt a flexible method.

Recall that, in Step 1, we collect a sufficient number of decision boundary points. For each tuple $(I_{\mathcal{P}}, B_{\mathcal{P}}) \in \mathcal{S}_{\theta}^{\mathcal{N}}$, denote by $m_{\mathcal{P}}$ the frequency that the tuple occurs in the collected decision boundary points. Suppose that the number of $m_{\mathcal{P}}$ where $m_{\mathcal{P}} > 1$ is N_{valid} . Then when at least $0.95 \times N_{\text{valid}}$ tuples $(I_{\mathcal{P}}, B_{\mathcal{P}}) \in \mathcal{S}_{\theta}^{\mathcal{N}}$ are in the set $\mathcal{S}_{\theta}^{\mathcal{N}}$, the extracted model $f_{\widehat{\theta}}$ is regarded as a candidate of the functionally equivalent model. Here, We call the ratio $0.95 \times \frac{N_{\text{valid}}}{|\mathcal{S}_{\theta}^{\mathcal{N}}|}$ the adaptive threshold.

Filtering by Weight Signs. After $\widehat{A}^{(i)}, \widehat{b}^{(i)}$ for $i \in \{1, \dots, k+1\}$ are obtained, we compute the matrices $\widehat{\mathcal{G}}^{(i)}$ and check whether the k signs, i.e., the sign of $\widehat{\mathcal{G}}_j^{(i)}, i \in \{1, \dots, k\}$ are consistent with the k guesses. If at least one sign is not consistent with the guess, the extracted model is not the functionally equivalent model.

Interestingly, except for handling wrong sign guesses, this method also shows high filtering effectiveness when the model activation patterns of the selected $n+1$ decision boundary points are not those required by our extraction attacks. This is not strange, since our extraction attack is designed for a specific set of model activation patterns. For wrong model activation patterns, whether the sign of $\widehat{\mathcal{G}}_j^{(i)}, i \in \{1, \dots, k\}$ is 1 or -1 is a random event.

Filtering by Prediction Matching Ratio. The above two filtering methods are effective, but we find that some functionally inequivalent models still escape from the filtering. Therefore, the third method is designed to perform the last filtering on extracted models surviving from the above two filtering methods. This method is based on the *prediction matching ratio*.

Prediction Matching Ratio. Randomly generate N_1 inputs, query the extracted model $f_{\widehat{\theta}}$ and the victim model f_{θ} . Suppose that the two models return the same hard-label for N_2 out of N_1 inputs. The ratio $\frac{N_2}{N_1}$ is called the prediction matching ratio.

According to Definition 1 and Definition 2, for a functionally equivalent model, the prediction matching ratio should be high, or even close to 100%. Note that many random inputs x and corresponding hard-label $z(f_{\theta}(x))$ are collected during the attack process (see Steps 1 and 2 in Sect. 4). Thus, we can exploit these inputs.

7 Experiments

Our model extraction attacks are evaluated on both untrained and trained neural networks. Concretely, we first perform experiments on untrained neural networks with diverse architectures and randomly generated parameters. Then, based on two typical benchmarking image datasets (i.e., MNIST, CIFAR10) in visual deep learning, we train a series of neural networks as classifiers and evaluate the model extraction attacks on these trained neural networks.

For convenience, denote by ‘ d_0 - d_1 - \dots - d_{k+1} ’ the victim model, where d_i is the dimension of each layer. For example, the symbol 1000-1 stands for a 0-deep neural network with an input dimension of 1000 and an output dimension of 1.

Partial Universal Experiment Settings. Some settings are used in all the following experiments. For k -deep neural network extraction attacks, in Step 1, we randomly generate 8×2^n pairs of starting point and moving direction, where $n = \sum_{i=1}^k d_i$ is the number of neurons. The prediction matching ratio is estimated over 10^6 random inputs.

7.1 Computing $(\varepsilon, 0)$ -Functional Equivalence

To quantify the degree to which a model extraction attack has succeeded, the method (i.e., error bounds propagation [4]) proposed by Carlini et al. is adopted to compute $(\varepsilon, 0)$ -functional equivalence.

Error bounds propagation. To compute $(\varepsilon, 0)$ -functional equivalence of the extracted neural network $f_{\hat{\theta}}$, one just needs to compare the extracted parameters (weights $\hat{A}^{(i)}$ and biases $\hat{b}^{(i)}$) to the real parameters (weights $A^{(i)}$ and biases $b^{(i)}$) and analytically derive an upper bound on the error when performing inference [4].

Before comparing the neural network parameters, one must ‘align’ them [4]. This involves two operations: (1) adjusting the order of the neurons in the network, i.e., the order of the rows or columns of $A^{(i)}$ and $b^{(i)}$, (2) adjusting the values of $A^{(i)}$ and $b^{(i)}$ to the theoretical one (see Eq. (19), Eq. (27), and Eq. (28)) obtained by the idealized model extraction attacks. This gives an aligned $\tilde{A}^{(i)}$ and $\tilde{b}^{(i)}$ from which one can analytically derive upper bounds on the error. Other details (e.g., propagating error bounds layer-by-layer) are the same as that introduced in [4], and not introduced again in this paper.

7.2 Experiments on Untrained Neural Networks

Table 1 summarizes the experimental results on different untrained neural networks which demonstrates the effectiveness of our model extraction attacks.

According to Appendix B, the computation complexity of our model extraction attack is about $\mathcal{O}\left(n \times 2^{n^2+n+k}\right)$, where n is the number of neurons. Thus, we limit the number of neurons, which does not influence the verification of our

Table 1. Experiment results on untrained k -deep neural networks.

Architecture	Parameters	ϵ	PMR	Queries	$(\epsilon, 0)$	$\max \theta - \hat{\theta} $
512-2-1	1029	10^{-12}	100%	$2^{19.35}$	$2^{-12.21}$	$2^{-16.88}$
		10^{-14}	100%	$2^{19.59}$	$2^{-19.84}$	$2^{-24.62}$
2048-4-1	8201	10^{-12}	99.98%	$2^{23.32}$	$2^{-3.77}$	$2^{-10.44}$
		10^{-14}	100%	$2^{23.51}$	$2^{-13.70}$	$2^{-17.75}$
25120-4-1	100489	10^{-14}	99.98%	$2^{26.42}$	$2^{-2.99}$	$2^{-14.67}$
		10^{-16}	100%	$2^{26.67}$	$2^{-13.01}$	$2^{-23.19}$
50240-2-1	100485	10^{-14}	99.99%	$2^{25.85}$	$2^{-7.20}$	$2^{-15.58}$
		10^{-16}	100%	$2^{26.31}$	$2^{-14.44}$	$2^{-22.67}$
32-2-2-1	75	10^{-12}	100%	$2^{17.32}$	$2^{-10.99}$	$2^{-14.78}$
		10^{-14}	100%	$2^{17.56}$	$2^{-18.21}$	$2^{-20.61}$
512-2-2-1	1035	10^{-12}	99.99%	$2^{21.39}$	$2^{-10.34}$	$2^{-14.01}$
		10^{-14}	100%	$2^{21.59}$	$2^{-14.17}$	$2^{-17.29}$
1024-2-2-1	2059	10^{-12}	99.99%	$2^{22.38}$	$2^{-6.10}$	$2^{-13.77}$
		10^{-14}	100%	$2^{22.49}$	$2^{-14.16}$	$2^{-20.38}$

ϵ : the precision used to find decision boundary points.

$\max|\theta - \hat{\theta}|$: the maximum extraction error of model parameters.

PMR: prediction matching ratio.

model extraction attack. Note that the number of parameters is not limited. All the attacks can be finished within several hours on a single core.

The results in Table 1 also support our argument in Remark 2. For the 2-deep neural networks (e.g., 32-2-2-1), when recovering the weights in layer 1, we require that only one neuron in layer 2 is active, instead of all the 2 neurons being active. Our extraction attacks also achieve good performance.

The Influence of the Precision ϵ . A smaller ϵ will make the returned point $x + s_{\text{slow}} \times \Delta$ (see Sect. 6.1) closer to the decision boundary, which helps reduce the extraction error of affine transformations. As a result, the model extraction attack is expected to perform better. For example, for the 1-deep neural network 2048-4-1, when ϵ decreases from 10^{-12} to 10^{-14} , the value ϵ (respectively, $\max|\theta - \hat{\theta}|$) decreases from $2^{-3.77}$ to $2^{-13.70}$ (respectively, from $2^{-10.44}$ to $2^{-17.75}$), which is a significant improvement.

At the same time, using a smaller precision ϵ does not increase the attack complexity significantly. According to Appendix B, the query complexity is about $\mathcal{O}\left(d_0 \times 2^n \times \log_2 \frac{1}{\epsilon}\right)$. Thus, decreasing ϵ has little influence on the query complexity. Look at the neural network 2048-4-1 again. When ϵ decreases from 10^{-12} to 10^{-14} , the number of queries only increases from $2^{23.32}$ to $2^{23.51}$. Besides, when n (i.e., the number of neurons) is large, ϵ almost does not influence the computation complexity, since ϵ only influences Steps 1 and 2 (see Sect. 4),

while the computation complexity is mainly determined by other steps (refer to Appendix B). When n is small, the practical runtime is determined by the query complexity, then decreasing ϵ also has little influence on the runtime.

Choosing an appropriate ϵ is simple. In our experiments, we find that a smaller ϵ should be used, when the prediction matching ratio estimated over 10^6 random inputs is not 100%, and the gap (e.g., 0.02%, see the third or fifth row) is not negligible.

7.3 Experiments on Trained Neural Networks

The MNIST and CIFAR10 Dataset. MNIST (respectively, CIFAR10) is one typical benchmarking dataset used in visual deep learning. It contains ten-class handwriting number gray images [12] (resp., real object images in a realistic environment [19]). Each of the ten classes, i.e., ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, and ‘9’ (resp., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), contains 28×28 pixel gray images (resp., 32×32 pixel RGB images), totaling 60000 (resp., 50000) training and 10000 (resp. 10000) testing images.

Neural Network Training Pipelines. When classifying different classes of objects, the decision boundary of trained neural networks will be different. To fully verify our model extraction attack, for MNIST (respectively, CIFAR10), we divide the ten classes into five groups and build a binary classification neural network for each group. All the neural networks share the same architecture d_0 -2-1, where $d_0 = 28 \times 28$ for MNIST (respectively, $32 \times 32 \times 3$ for CIFAR10). On the MNIST and CIFAR10 datasets, we perform a standard rescaling of the pixel values from $0 \cdots 255$ to $0 \cdots 1$. For the model training, we choose typical settings (the loss is the cross-entropy loss; the optimizer is standard stochastic gradient descent; batch size 128). The first four columns of Table 2 summarize a detailed description of the neural networks to be attacked in this section.

Experiment Results. The last four columns of Table 2 summarize the experiment results. Our extraction attack still achieves good performance when an appropriate precision ϵ is used, which further verifies its effectiveness.

The experimental results presented in Table 1 and Table 2 show that the attack performance (i.e., the value of ϵ and $\max|\theta - \hat{\theta}|$) is related to the precision ϵ and the properties of the decision boundary. However, we do not find a clear quantitative relationship between the attack performance and the precision ϵ (or some unknown properties of the decision boundary). Considering that the unknown quantitative relationships do not influence the verification of the model extraction attack, we leave the problem of exploring the unknown relationships as a future work.

Table 2. Experiment results on neural networks trained on MNIST or CIFAR10.

task	architecture	accuracy	parameters	ϵ	Queries	$(\epsilon, 0)$	$\max \theta - \widehat{\theta} $
'0' vs '1'	784-2-1	0.9035	1573	10^{-12}	$2^{20.11}$	$2^{-16.39}$	$2^{-17.85}$
				10^{-14}	$2^{20.32}$	$2^{-20.56}$	$2^{-22.81}$
'2' vs '3'	784-2-1	0.8497	1573	10^{-12}	$2^{20.11}$	$2^{-7.00}$	$2^{-7.80}$
				10^{-14}	$2^{20.32}$	$2^{-14.32}$	$2^{-15.06}$
'4' vs '5'	784-2-1	0.8570	1573	10^{-12}	$2^{20.02}$	$2^{-8.47}$	$2^{-8.82}$
				10^{-14}	$2^{20.32}$	$2^{-15.62}$	$2^{-15.81}$
'6' vs '7'	784-2-1	0.9290	1573	10^{-12}	$2^{20.11}$	$2^{-7.02}$	$2^{-7.93}$
				10^{-14}	$2^{20.32}$	$2^{-12.00}$	$2^{-12.91}$
'8' vs '9'	784-2-1	0.9501	1573	10^{-12}	$2^{20.11}$	$2^{-10.58}$	$2^{-11.62}$
				10^{-14}	$2^{20.32}$	$2^{-19.63}$	$2^{-21.72}$
airplane vs automobile	3072-2-1	0.8120	6149	10^{-12}	$2^{22.08}$	$2^{-4.84}$	$2^{-7.48}$
				10^{-14}	$2^{22.29}$	$2^{-12.41}$	$2^{-15.20}$
bird vs cat	3072-2-1	0.6890	6149	10^{-12}	$2^{22.07}$	$2^{-8.37}$	$2^{-9.80}$
				10^{-14}	$2^{22.29}$	$2^{-12.27}$	$2^{-14.73}$
deer vs dog	3072-2-1	0.6870	6149	10^{-12}	$2^{22.01}$	$2^{-9.55}$	$2^{-13.25}$
				10^{-14}	$2^{22.22}$	$2^{-13.19}$	$2^{-15.82}$
frog vs horse	3072-2-1	0.8405	6149	10^{-12}	$2^{22.08}$	$2^{-9.56}$	$2^{-10.71}$
				10^{-14}	$2^{22.29}$	$2^{-13.58}$	$2^{-15.58}$
ship vs truck	3072-2-1	0.7995	6149	10^{-12}	$2^{22.08}$	$2^{-8.63}$	$2^{-8.90}$
				10^{-14}	$2^{22.29}$	$2^{-12.95}$	$2^{-13.02}$

$\max|\theta - \widehat{\theta}|$: the maximum extraction error of model parameters.

accuracy: classification accuracy of the victim model f_θ .

for saving space, prediction matching ratios are not listed.

8 Conclusion

In this paper, we have studied the model extraction attack against neural network models under the hard-label setting, i.e., the adversary only has access to the most likely class label corresponding to the raw output of neural network models. We propose new model extraction attacks that theoretically achieve functionally equivalent extraction. Practical experiments on numerous neural network models have verified the effectiveness of the proposed model extraction attacks. To the best of our knowledge, this is the first time to prove with practical experiments that it is possible to achieve functionally equivalent extraction against neural network models under the hard-label setting.

The future work will mainly focus on the following aspects:

- The (computation and query) complexity of our model extraction attack remains high, which limits the application to neural networks with a large number of neurons. Reducing the complexity is an important problem.

- In this paper, to recover the weight vector of the j -th neuron in layer i , we require that in layer i , only the j -th neuron is active. However, such a model activation pattern may not occur in some cases. Then how to recover the weight vector of this neuron based on other model activation patterns would be a vital step towards better generality.
- Explore possible quantitative relationships between the precision ϵ (or some unknown properties of the decision boundary) and ε (or $\max|\theta - \hat{\theta}|$).
- Extend the extraction attack to the case of vector outputs, i.e., the output dimensionality exceeds 1.
- Develop extraction attacks against other kinds of neural network models.

Acknowledgments. We would like to thank Adi Shamir for his guidance. We would like to thank the anonymous reviewers for their detailed and helpful comments. This work was supported by the National Key R&D Program of China (2018YFA0704701, 2020YFA0309705), Shandong Key Research and Development Program (2020ZLYS09), the Major Scientific and Technological Innovation Project of Shandong, China (2019JZZY010133), the Major Program of Guangdong Basic and Applied Research (2019B030302008), the Tsinghua University Dushi Program, and the Ministry of Education in Singapore under Grant RG93/23. Y. Chen was also supported by the Shuimu Tsinghua Scholar Program.

A Proof of Lemma 1

We prove Lemma 1 by Mathematical Induction.

Proof. When $i = 2$, according to Lemma 1, the extracted weight vector $\hat{A}_j^{(2)}, j \in \{1, \dots, d_2\}$ should be

$$\begin{aligned} \hat{A}_j^{(2)} &= \left[\frac{w_{j,1}^{(2)} \times \left| \sum_{v=1}^{d_0} w_{1,v}^{(1)} C_{v,1}^{(0)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|}, \dots, \frac{w_{j,d_1}^{(2)} \times \left| \sum_{v=1}^{d_0} w_{d_1,v}^{(1)} C_{v,1}^{(0)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \right] \\ &= \left[\frac{w_{j,1}^{(2)} \times \left| w_{1,1}^{(1)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|}, \dots, \frac{w_{j,d_1}^{(2)} \times \left| w_{d_1,1}^{(1)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \right]. \end{aligned} \tag{33}$$

Note that $C_{1,1}^{(0)} = 1$ and $C_{v,1}^{(0)} = 0$ for $v \in \{2, \dots, d_0\}$.

Besides, we have

$$\begin{aligned} C^{(1)} &= I_{\mathcal{P}}^{(1)} A^{(1)} = A^{(1)} = \left[A_1^{(1)}, \dots, A_{d_0}^{(1)} \right], \\ \hat{C}^{(1)} &= I_{\mathcal{P}}^{(1)} \hat{A}^{(1)} = \hat{A}^{(1)} = \left[\frac{A_1^{(1)}}{\left| w_{1,1}^{(1)} \right|}, \dots, \frac{A_{d_0}^{(1)}}{\left| w_{d_0,1}^{(1)} \right|} \right]. \end{aligned} \tag{34}$$

where $A_v^{(1)} = \left[w_{v,1}^{(1)}, \dots, w_{v,d_0}^{(1)} \right]$, $C_{v,u}^{(1)} = w_{v,u}^{(1)}$ and $\hat{C}_{v,u}^{(1)} = \frac{w_{v,u}^{(1)}}{\left| w_{v,1}^{(1)} \right|}$.

Look at the system of linear equations presented in Eq. (26). Now, the system of linear equations is transformed into

$$\begin{cases} \sum_{v=1}^{d_1} \widehat{w}_{j,v}^{(2)} \widehat{C}_{v,1}^{(1)} = \frac{\sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)}}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \\ \vdots \\ \sum_{v=1}^{d_1} \widehat{w}_{j,v}^{(2)} \widehat{C}_{v,d_0}^{(1)} = \frac{\sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,d_0}^{(1)}}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \end{cases} \quad (35)$$

when $d_0 \geq d_1$, by solving the system, it is expected to obtain

$$\widehat{A}_j^{(2)} = \left[\frac{w_{j,1}^{(2)} \left| w_{1,1}^{(1)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|}, \dots, \frac{w_{j,d_1}^{(2)} \left| w_{d_1,1}^{(1)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \right], \quad (36)$$

which is consistent with the expected value in Eq. (33).

Next, consider the recovery of the weight vector of the j -th neuron in layer i , and assume that the weights $\widehat{A}^{(1)}, \dots, \widehat{A}^{(i-1)}$ as shown in Lemma 1 have been obtained. As a result, we have

$$\begin{aligned} C_j^{(i-2)} &= A_j^{(i-2)} A^{(i-3)} \dots A^{(1)}, \quad C_j^{(i-1)} = A_j^{(i-1)} A^{(i-2)} \dots A^{(1)}, \\ \widehat{C}_j^{(i-2)} &= \widehat{A}_j^{(i-2)} \widehat{A}^{(i-3)} \dots \widehat{A}^{(1)} = \frac{C_v^{(i-2)}}{\left| \sum_{v=1}^{d_{i-3}} w_{j,v}^{(i-2)} C_{v,1}^{(i-3)} \right|}, \\ \widehat{C}_j^{(i-1)} &= \widehat{A}_j^{(i-1)} \widehat{A}^{(i-2)} \dots \widehat{A}^{(1)} = \frac{C_v^{(i-1)}}{\left| \sum_{v=1}^{d_{i-2}} w_{j,v}^{(i-1)} C_{v,1}^{(i-2)} \right|}. \end{aligned} \quad (37)$$

Now, the system of linear equations in Eq. (26) is transformed into

$$\begin{cases} \sum_{u=1}^{d_{i-1}} \widehat{w}_{j,u}^{(i)} \frac{C_{u,1}^{(i-1)}}{\left| \sum_{v=1}^{d_{i-2}} w_{u,v}^{(i-1)} C_{v,1}^{(i-2)} \right|} = \frac{\sum_{u=1}^{d_{i-1}} w_{j,u}^{(i)} C_{u,1}^{(i-1)}}{\left| \sum_{u=1}^{d_{i-1}} w_{j,u}^{(i)} C_{u,1}^{(i-1)} \right|} \\ \vdots \\ \sum_{u=1}^{d_{i-1}} \widehat{w}_{j,u}^{(i)} \frac{C_{u,d_0}^{(i-1)}}{\left| \sum_{v=1}^{d_{i-2}} w_{u,v}^{(i-1)} C_{v,1}^{(i-2)} \right|} = \frac{\sum_{u=1}^{d_{i-1}} w_{j,u}^{(i)} C_{u,d_0}^{(i-1)}}{\left| \sum_{u=1}^{d_{i-1}} w_{j,u}^{(i)} C_{u,1}^{(i-1)} \right|} \end{cases} \quad (38)$$

When $d_0 \geq d_{i-1}$, by solving this system, it is expected to obtain

$$\begin{aligned} \widehat{A}_j^{(i)} &= \left[\widehat{w}_{j,1}^{(i)}, \dots, \widehat{w}_{j,d_{i-1}}^{(i)} \right] \\ &= \left[\frac{w_{j,1}^{(i)} \times \left| \sum_{v=1}^{d_{i-2}} w_{1,v}^{(i-1)} C_{v,1}^{(i-2)} \right|}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \dots, \frac{w_{j,d_{i-1}}^{(i)} \times \left| \sum_{v=1}^{d_{i-2}} w_{d_{i-1},v}^{(i-1)} C_{v,1}^{(i-2)} \right|}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|} \right], \end{aligned}$$

which is consistent with the expected value in Eq. (27).

B Complexity of Hard-Label Model Extraction Attacks

For the k -deep neural network extraction attack, its complexity is composed of two parts: Oracle query complexity and computation complexity. Suppose that the number of neurons is $n = \sum_{i=1}^k d_i$. Its input size, i.e., the size of x is d_0 . And k is the number of hidden layers. The precision adopted by binary search is ϵ (refer to Sect. 6.1).

Oracle Query Complexity. For the k -deep neural network extraction attack, we only query the Oracle in Steps 1 and 2 (see Sect. 4).

In Step 1, if $c_n \times 2^n$ decision boundary points are collected, then the number of queries to the Oracle is $c_\epsilon \times c_n \times 2^n$, where c_ϵ is a factor determined by the precision ϵ , and c_n is a small factor defined by the attacker. In Step 2, for each decision boundary point x collected in Step 1, to recover the corresponding affine transformation (i.e., $\Gamma_{\mathcal{P}}$ and $B_{\mathcal{P}}$), we need to collect another $d_0 - 1$ decision boundary points. Therefore, the times of querying the Oracle in this step is $c_\epsilon \times c_n \times 2^n \times (d_0 - 1)$. Based on the above analysis, the Oracle query complexity of our k -deep neural network extraction attack is $c_\epsilon \times c_n \times 2^n \times d_0$. Note that c_ϵ is proportional to $\log_2^{\frac{1}{\epsilon}}$. Thus, the query complexity is about $\mathcal{O}\left(d_0 \times 2^n \times \log_2^{\frac{1}{\epsilon}}\right)$.

Computation Complexity. For the k -deep neural network extraction attack, when n is large, most computations are occupied by recovering neural network parameters, i.e., Steps 3 and 4 (see Sect. 4). Suppose that there are $N \leq c_n \times 2^n$ decision boundary points used to recover neural network parameters after filtering duplicate affine transformations in Step 2.

In Step 3, to recover the weight vector of the j -th neuron in layer i where $i \in \{2, \dots, k+1\}$, we need to solve a system of linear equations. For convenience, let us ignore the difference in the sizes of the different systems of linear equations. Then, to recover all the weights $A^{(i)}$, a total of $n + 1 - d_1 = \sum_{i=2}^{k+1} d_i$ systems of linear equations need to be solved. In Step 4, to recover all the biases $b^{(1)}$, only one system of linear equations needs to be solved. Therefore, to obtain an extracted model, we need to solve $n + 2 - d_1$ systems of linear equations.

There are two loops in the extraction attack. First, we need to select $n + 1$ out of N decision boundary points each time. More concretely, to recover the weights $A^{(i)}$ in layer i , we choose d_i decision boundary points. Then the number (denoted by l_1) of possible cases is

$$l_1 = \binom{N}{d_1} \times \binom{N-d_1}{d_2} \times \dots \times \binom{N-\sum_{i=1}^{k-1} d_i}{d_k} \times \binom{N-n}{1} \approx N^{n+1}, \text{ for } N \gg n.$$

Second, we need to guess k signs when recovering all the weights, i.e., there are 2^k cases.

Thus, the computation complexity is about $\mathcal{O}(l_1 \times 2^k \times (n + 2 - d_1))$. When an appropriate precision ϵ (i.e., ϵ is small) is adopted, we have $N \approx H < 2^n$, where H is the number of possible model activation patterns (refer to Lemma 2). Then, we further have

$$l_1 \times 2^k \times (n + 2 - d_1) \approx N^{n+1} \times 2^k \times n \approx n \times 2^{n(n+1)+k}. \quad (39)$$

Thus, the computation complexity is about $\mathcal{O}(n \times 2^{n^2+n+k})$.

C Extraction on 1-Deep Neural Networks

The parameters of the extracted 1-deep neural network are as follows.

$$\begin{aligned}
 \widehat{A}_i^{(1)} &= [\widehat{w}_{i,1}^{(1)}, \dots, \widehat{w}_{i,d_0}^{(1)}] = \left[\frac{w_{i,1}^{(1)}}{|w_{i,1}^{(1)}|}, \dots, \frac{w_{i,d_0}^{(1)}}{|w_{i,1}^{(1)}|} \right], \quad i \in \{1, \dots, d_1\}, \\
 \widehat{b}^{(1)} &= [\widehat{b}_1^{(1)}, \dots, \widehat{b}_{d_1}^{(1)}] = \left[\frac{b_1^{(1)}}{|w_{1,1}^{(1)}|}, \dots, \frac{b_{d_1}^{(1)}}{|w_{d_1,1}^{(1)}|} \right], \\
 \widehat{A}^{(2)} &= [\widehat{w}_1^{(2)}, \dots, \widehat{w}_{d_1}^{(2)}] = \left[\frac{w_1^{(2)} |w_{1,1}^{(1)}|}{|\sum_{i=1}^{d_1} w_i^{(2)} w_{i,1}^{(1)}|}, \dots, \frac{w_{d_1}^{(2)} |w_{d_1,1}^{(1)}|}{|\sum_{i=1}^{d_1} w_i^{(2)} w_{i,1}^{(1)}|} \right], \\
 \widehat{b}^{(2)} &= \frac{b^{(2)}}{|\sum_{i=1}^{d_1} w_i^{(2)} w_{i,1}^{(1)}|}.
 \end{aligned} \tag{40}$$

Figure 4 shows a diagram of a victim model (2-2-1) and the extracted model.

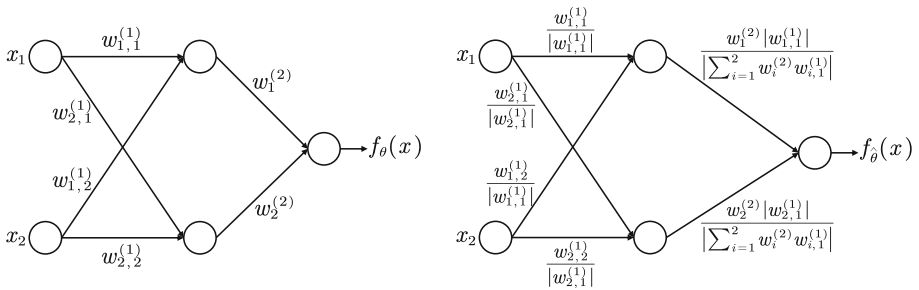


Fig. 4. Left: the victim model f_θ . Right: the extracted model $f_{\widehat{\theta}}$.

References

1. Batina, L., Bhasin, S., Jap, D., Picek, S.: CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019. pp. 515–532. USENIX Association
2. Blum, A., Rivest, R.L.: Training a 3-node neural network is np-complete. In: Hanson, S.J., Remmele, W., Rivest, R.L. (eds.) Machine Learning: From Theory to Applications - Cooperative Research at Siemens and MIT. LNCS, vol. 661, pp. 9–28. Springer (1993)

3. Canales Martinez, I.A., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Satpute, N., Shamir, A.: Polynomial time cryptanalytic extraction of neural network models. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024. LNCS, vol. 14653, pp. 3–33. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58734-4_1
4. Carlini, N., Jagielski, M., Mironov, I.: Cryptanalytic extraction of neural network models. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 189–218. Springer
5. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: SP 2017. pp. 39–57. IEEE Computer Society
6. Dubey, S.R., Singh, S.K., Chaudhuri, B.B.: Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing* **503**, 92–108 (2022)
7. Fefferman, C.: Reconstructing a neural net from its output. *Revista Matematica Iberoamericana* **10**, 507–555 (1994)
8. Galstyan, A., Cohen, P.R.: Empirical comparison of "hard" and "soft" label propagation for relational classification. In: Blockeel, H., Ramon, J., Shavlik, J.W., Tadepalli, P. (eds.) ILP 2007. LNCS, vol. 4894, pp. 98–111. Springer
9. Ganju, K., Wang, Q., Yang, W., Gunter, C.A., Borisov, N.: Property inference attacks on fully connected neural networks using permutation invariant representations. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) CCS 2018. pp. 619–633. ACM
10. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural network. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) NeurIPS 2015. pp. 1135–1143
11. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: Capkun, S., Roesner, F. (eds.) USENIX Security 2020. pp. 1345–1362. USENIX Association
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998)
13. Long, C., Collins, R., Swears, E., Hoogs, A.: Deep neural networks in fully connected CRF for image labeling with social network metadata. In: WACV 2019. pp. 1607–1615. IEEE
14. Lowd, D., Meek, C.: Adversarial learning. In: Grossman, R., Bayardo, R.J., Bennett, K.P. (eds.) SIGKDD 2005. pp. 641–647. ACM
15. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Fürnkranz, J., Joachims, T. (eds.) ICML, 2010. pp. 807–814. Omnipress
16. Oliyynyk, D., Mayer, R., Rauber, A.: I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Comput. Surv.* **55**(14s), 324:1–324:41 (2023)
17. Perazzi, F., Wang, O., Gross, M.H., Sorkine-Hornung, A.: Fully connected object proposals for video segmentation. In: ICCV 2015. pp. 3227–3234. IEEE Computer Society
18. Rolnick, D., Kording, K.P.: Reverse-engineering deep relu networks. In: ICML 2020. Proceedings of Machine Learning Research, vol. 119, pp. 8178–8187. PMLR
19. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **30**(11), 1958–1970 (2008)
20. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 601–618. USENIX Association



Analysis on Sliced Garbling via Algebraic Approach

Taechan Kim^(✉)

Seoul, Korea
tckim1458@gmail.com

Abstract. Recent improvements to garbled circuits are mainly focused on reducing their size. The state-of-the-art construction of Rosulek and Roy (Crypto 2021) requires 1.5κ bits for garbling AND gates in the free-XOR setting. This is below the previously proven lower bound 2κ in the linear garbling model of Zahur, Rosulek, and Evans (Eurocrypt 2015). Recently, Ashur, Hazay, and Satish (eprint 2024/389) proposed a scheme that requires $4/3\kappa + O(1)$ bits for garbling AND gates. Precisely they extended the idea of *slicing* introduced by Rosulek and Roy to garble 3-input gates of the form $g(u, v, w) := u(v + w)$. By setting $w = 0$, it can be used to garble AND gates with the improved communication costs. However, in this paper, we observe that the scheme proposed by Ashur, Hazy, and Satish leaks information on the permute bits, thereby allowing the evaluator to reveal information on the private inputs. To be precise, we show that in their garbling scheme, the evaluator can compute the bits α and $\beta + \gamma$, where α , β , and γ are the private permute bits of the input labels A , B , and C , respectively.

1 Introduction

Garbled Circuits (GC) are one of major techniques for secure two-party computation, which allows two mistrusting parties to jointly compute functions on their private inputs while revealing only the outputs of the functions and nothing else. Since their concept was first introduced by Yao [16], one line of recent research [4, 7, 11–15, 17] has been dedicated to reducing the size of the garbled circuit ciphertexts that should be sent from one party, the garbler, to the other party, the evaluator.

The current state-of-the-art construction for garbled circuits is due to Rosulek and Roy [15] (dubbed as RR21 throughout the paper), where they consider a gate-by-gate garbling of Boolean circuits expressed using XOR and AND gates. In their scheme, the size of the garbled AND gates is 1.5κ bits (κ is the security parameter), while no communication is required for XOR gates.

In this work, we present an attack on a new garbling scheme proposed by Ashur, Hazay, and Satish [1]. Concurrently to our work, we noticed that Fan, Lu, and Zhou [6] also described an attack on their scheme.

T. Kim—Independent Researcher.

Their result surpassed the previous lower bound (2κ bits for AND gates with free-XOR) for the size of garbled circuits, which is obtained in a model called *linear garbling* defined by Zahur, Rosulek, and Evans [17]. Their optimization was made possible by a new technique, called *slicing-and-dicing*, that is beyond the definition of the linear garbling model.

Following to the previous works, Ashur, Hazay, and Satish [1] recently proposed a garbling scheme that garbles AND gates requiring communication costs of only $4/3\kappa + O(1)$ bits, thus improving upon the previous state-of-the-art construction. Their core idea is to extend the slicing-and-dicing technique by Rosulek and Roy. Precisely, they suggested to garble 3-input gates of the form $g(u, v, w) := u(v + w)$ (where the function is defined over the binary field \mathbb{F}_2) instead of directly garbling AND gates. They also suggested to slice the input labels into 3 pieces, whereas the RR21 construction uses 2-sliced input labels. By setting $w = 0$ in $g(u, v, w)$, one can use their garbling scheme to garble AND gates.

Our Contributions. However, in this paper, we show that their garbling scheme leaks private information on inputs, thereby jeopardizing the security guarantee that should be satisfied in the garbling scheme. Precisely, we prove that the evaluator can compute the bits α and $\beta + \gamma$, where α , β , and γ are private permuted bits of the input labels A , B , and C , respectively.¹ The permuted bits are used to mask the private inputs u , v , and w of the gate g , thus it should not be revealed to the evaluator to satisfy the privacy property of the garbling scheme.

Previous Works. Before describing our techniques, we briefly review prior approaches. Since its introduction by Yao [16], the core idea behind garbled circuits has centered on *encoding the truth table of a function*. For a function $g : \{0, 1\}^n \rightarrow \{0, 1\}$, the truth table contains $N = 2^n$ rows. Each i -th input is assigned an input label, represented as a κ -bit string, depending on the input's truth value. Likewise, the output is assigned either an output label. The garbling of g is performed by encrypting the output label corresponding to the value $v = g(u_1, \dots, u_n)$, using the n input labels corresponding to u_i 's as encryption keys. This naive approach results in a garbled function of size $N\kappa$ bits.

To fix idea, let us focus on garbling 2-input gates where each input wire has labels (A_0, A_1) and (B_0, B_1) . From now on we assume the point-and-permute technique [4] is applied with the free-XOR setting. Precisely, let A_α and B_β be the labels corresponding to the logical value 0 on the respective input wire, where the masking bits α and β (a.k.a. the permuted bits) are secretly known by the garbler. Equivalently, $A_{u+\alpha}$ and $B_{v+\beta}$ correspond to the values u and v respectively, where the addition of the subscripts is over \mathbb{F}_2 . In the point-and-permute technique, if the evaluator holds one of $\{A_0, A_1\}$ and $\{B_0, B_1\}$, say A_x

¹ Each of the input labels is a κ -bit string that is assigned to each of the inputs of the gate depending on their logical values. The input labels A , B , and C correspond to the input u , v , and w , respectively.

and B_y , then she would know their subscripts x and y (a.k.a. color bits and they are typically given as the least significant bit of labels). Here, we see that only the masked bits $x = u + \alpha$ and $y = v + \beta$ are revealed to the evaluator and the truth values u and v are still hidden as the random mask α and β are only known by the garbler.

When garbling a Boolean circuit, a common technique is to decompose the circuit into a series of AND and XOR gates and apply a gate-by-gate garbling method. With the free-XOR technique [12], all wire labels share a global offset Δ , meaning $A_0 + A_1 = B_0 + B_1 = \Delta$, allowing XOR gates to be garbled for free. So, recent optimizations have focused on reducing the size of the ciphertexts required for garbling AND gates. Yao’s original circuit requires 4κ bits per AND gate, but this was later reduced to 2κ bits using the half-gate garbling technique [17]. The current state-of-the-art achieves a size of $1.5\kappa + O(1)$ bits, as shown by Rosulek and Roy [15].

A key observation in [15] is that the encoded truth table can be viewed as a system of linear equations. To illustrate, consider Yao’s garbled circuit under the free-XOR setting. The encoded truth table for an AND gate, consisting of four rows, is represented as follows:

$$\begin{aligned} C &= G_{0,0} + H(A_0, B_0) \\ C &= G_{0,1} + H(A_0, B_1) \\ C &= G_{1,0} + H(A_1, B_0) \\ C + \Delta &= G_{1,1} + H(A_1, B_1), \end{aligned}$$

where $G_{i,j}$ are the four ciphertexts corresponding to combinations of input labels (A_i, B_j) , C is the output label corresponding to the truth value 0, and we assume the permute bits $(\alpha, \beta) = (1, 1)$. This can be rearranged into a system of four linear equations:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C \\ G_{0,0} \\ G_{0,1} \\ G_{1,0} \\ G_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} H(A_0, B_0) \\ H(A_0, B_1) \\ H(A_1, B_0) \\ H(A_1, B_1) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \Delta.$$

Here, given the input labels and the global offset, the garbler solves for the variables on the left-hand side. Notably, the matrix on the right-hand side has rank 4, and the number of ciphertexts required is determined by this rank. While the example suggests that four ciphertexts are necessary, one of the ciphertexts can be set as a zero string using a row-reduction technique [4]. This shows that 4 degrees of freedom are used to set the output label and three ciphertexts on the left-hand side.

In the half-gate garbling technique, random oracle queries take the form of $H(A_i)$ and $H(B_j)$ instead of $H(A_i, B_j)$, which results in a lower-rank garbling equation and, therefore, fewer ciphertexts.

In summary, constructing an improved garbled circuit involves finding appropriate linear equations of lower rank using the input labels and the global offset.

To further reduce ciphertext size, Rosulek and Roy [15] introduced new types of random oracle queries, such as $H(A_i + B_j)$, alongside the existing $H(A_i)$ and $H(B_j)$ queries. Additionally, they split the output labels into two parts and considered eight (i.e., 2×4) linear equations, where each row represents the left or right half of the output labels based on the four possible input label combinations. This approach allowed to obtain smaller ciphertexts, but required an exhaustive search to finalize the garbling equation.

The recent approach by [1] builds on this idea of a linear algebraic representation of garbling equations. They extended the techniques from [15] to 3-input gates and demonstrated how to garble a gate of the form $g(u, v, w) := u(v + w)$. While they explained how to apply the linear algebraic approach to this gate, an explicit formula has not yet been provided.

More details can be found in the subsequent sections.

Our Techniques. Previous works, such as [1, 15], viewed garbling schemes as systems of linear equations. The core idea of this paper is based on a novel observation by [2], which reformulates these linear systems in garbling schemes as algebraic equations. In this approach, the color bits x and y are treated as variables in \mathbb{F}_2 , and the garbling equation is expressed as polynomials over \mathbb{F}_{2^k} , with x and y as variables.

Let us revisit Yao's circuit as a concrete example. Using the algebraic framework of [2], the system of linear equations can be reformulated as follows:

$$\begin{aligned} & C + (x + 1)(y + 1)G_{0,0} + (x + 1)yG_{0,1} + x(y + 1)G_{1,0} + xyG_{1,1} \\ &= (x + 1)(y + 1)H_{0,0} + (x + 1)yH_{0,1} + x(y + 1)H_{1,0} + xyH_{1,1} + (x + \alpha)(y + \beta)\Delta, \end{aligned}$$

where $H_{i,j} := H(A_i, B_j)$. This shows that the garbling equation in Yao's circuit can be represented using quadratic polynomials, with coefficients on the right-hand side determined by the input labels and the global offset.

More generally, previous garbling schemes can be expressed in the following form:

$$C + g(x + \alpha, y + \beta)\Delta = \mathcal{F}(x, y),$$

where \mathcal{F} is a function that takes the color bits x and y as inputs and g is a target function to be garbled.

This simple reformulation allows garbling schemes to be represented in a more compact and elegant form, making it easier to identify a suitable function \mathcal{F} that yields a correct garbling scheme. A major challenge in the constructions of [15] and [1] was to find the *control matrices* required to complete the function \mathcal{F} for a valid garbling scheme.

While the work of [2] primarily focuses on garbling 2-input AND gates, we observe that this algebraic perspective can be extended to garbling gadgets with an arbitrary number of inputs and degrees. This representation allows us to derive an *explicit formula* for the garbling equation in the construction by Ashur et al. Once this formula is established, it becomes evident that this construction leaks the permute bits.

As a side result, we derive an impossibility result, showing that garbling gates of degree greater than 2 is not feasible if the only allowed queries to the oracle are restricted to linear functions of the input labels. This result aligns with Theorem 3 from Ashur et al. [1]. For gates of degree 2, we provide sufficient conditions under which our attack can succeed. This also offers guidance on *how not to* design garbling schemes to reduce communication costs.

Limitations. We remark that our analysis primarily considers garbling schemes within the linear garbling model, as defined by [17]. In essence, this model permits only linear operations and queries to the random oracle. While the state-of-the-art construction [15] slightly deviates from the original definition of the linear garbling model, it still adheres to the principle of using only linear operations and random oracle queries, with the exception that wire labels are split into two parts, each obtained through linear operations and random oracle queries. In our analysis, we extend this by allowing wire labels to be split into multiple parts, though they are still derived exclusively through linear operations and random oracle queries. Additionally, we assume, as in previous approaches [1, 15], that inputs to the random oracle are formed as linear combinations of the input labels. We exclude non-linear queries, as they appear to increase the number of ciphertexts due to the higher degree in the garbling equation.

It is also important to note that our focus does not extend to constructions that operate outside the linear garbling model. For example, we do not consider constructions based on one-hot garbling [8, 9] or those relying on DCR assumptions [3].

Concurrent Works. Concurrently to our work, Fan, Lu, and Zhou [6] also described an attack on the scheme by Ashur, Hazay, and Satish [1]. In their work, they described how the permute bit α can be revealed using the linear-algebraic representation of garbling schemes when the evaluator's color bits are $(0, 0, 0)$. On the other hand, with our algebraic approach, we can provide more general results: we show that the evaluator can reveal not only the permute bit α , but also the value $\beta + \gamma$, regardless of the evaluator's given color bits.

2 Preliminaries

Throughout the paper, we will work over finite fields \mathbb{K} of characteristic 2 and the bivariate polynomial ring $\mathbb{K}[x, y]$. We write $x + y/xy$ for Boolean operations XOR/AND of $x, y \in \mathbb{F}_2$, respectively. We denote a vector and its entries as $\vec{v} = (v_1, \dots, v_n)$. Matrices are written in the bold capital characters such as \mathbf{M} .

2.1 Garbling Schemes

We use the garbling scheme abstraction introduced by Bellare, Hoang, and Rogaway [5]. In particular, as in [17], we concentrate on garbling circuits rather than garbling any form of computation. A garbling scheme consists of the following algorithms:

- **Gb**: On input 1^κ and a Boolean circuit f , outputs (F, e, d) , where F is a *garbled circuit*, e is encoding information, d is decoding information.
- **En**: On input (e, x) , where e is as above and x is an input suitable for f , outputs a *garbled input* X .
- **Ev**: On input (F, X) , outputs a *garbled output* Y .
- **De**: On input (d, Y) , returns an output y .

Correctness. A garbling scheme defined as above is *correct*, if $(F, e, d) \leftarrow \text{Gb}(1^\kappa, f)$, $\text{De}(d, \text{Ev}(F, \text{En}(e, x))) = f(x)$ holds all but negligible probability.

Privacy. Informally, we say a garbling scheme satisfies *privacy*, if (F, X, d) reveals no information about x other than $f(x)$. In our discussion, it is enough to consider the privacy property. For further details on other security properties such as *obliviousness* and *authenticity*, refer to Bellare, Hoang, and Rogaway [5].

3 Algebraic Understanding of Garbling Schemes

In [15], garbling schemes were interpreted as systems of linear equations. Building on this idea, they were able to devise an efficient garbling scheme that improved upon previous state-of-the-art constructions. In this section, we review the reformulation introduced by [2], which offers an algebraic perspective on existing garbling schemes.

3.1 Review on Existing Schemes

For now, we focus on a gate g with input wires a, b and output wire c . If g is the AND gate, we have $c = g(a, b) = a \cdot b$ for $a, b \in \mathbb{F}_2$. From now on we assume the point-and-permute techniques [4].

Notations.

- (*Permute bits*) The values α and β are secret permute bits that are only known by the garbler.
- (*Input labels*) $A_\alpha, B_\beta \in \{0, 1\}^\kappa$ are wire labels corresponding to the **false** value on input wires a and b , respectively.
- (*Output label*) $C \in \{0, 1\}^\kappa$ represents the output wire label corresponding to the **false** value on the output wire c .
- (*Color bits*) When A_x and B_y are held by the evaluator, we assume that the subscripts, x and y , referred to as color bits, are known by the evaluator.
- (*Free-XOR*) For each wire label W , it holds $W_0 + W_1 = \Delta$, with the addition being performed over \mathbb{F}_2 . Here $\Delta \in \{0, 1\}^\kappa$ is a global offset.
- (*Sliced labels*) Given an integer s , we represent the wire label $W = W^1 \parallel \dots \parallel W^s$ as $\mathbf{W} = (W^1, \dots, W^s)$, where each W^i is κ/s -bits. If $s = 1$, we simply write W instead of \mathbf{W} .

With the above notations, the wire labels A_x and B_y correspond to the logical values $u := x + \alpha$ and $v := y + \beta$, where the addition is computed over \mathbb{F}_2 .

As a matter of mathematical conventions, we abstract strings in $\{0, 1\}^\kappa$ as fields elements in \mathbb{F}_{2^κ} . However, the field structures that we are only interested in are additions and multiplications by \mathbb{F}_2 (no multiplications by full field elements are required).

Yao's Garbled Circuits. We describe the classical Yao's garbled circuits from the algebraic perspective. In Yao's circuit, the garbler generates the ciphertexts $G_{x,y}$ for each $(x, y) \in \mathbb{F}_2 \times \mathbb{F}_2$ in a way that the following equation holds:

$$C + (x + \alpha)(y + \beta)\Delta = G_{x,y} + H(A_x, B_y). \tag{1}$$

In other words, decryption of $G_{x,y}$ under the key $H(A_x, B_y)$ yields to the value $C + \Delta$, the output wire label corresponding to the logical value 1, if and only if $(x + \alpha)(y + \beta) = 1$.

Let us consider $G_{x,y}$ and $H_{x,y} := H(A_x, B_y)$ as functions in variables x, y with their values in \mathbb{F}_{2^κ} . Particularly, we only consider x, y that take values in \mathbb{F}_2 , then we may write the functions as formal summations using group algebra:

$$\begin{aligned} \mathbb{T}_\kappa &:= \mathbb{F}_{2^\kappa} [\mathbb{F}_2[x, y]/(x^2 + x, y^2 + y)] \\ &= \left\{ \sum_f a_f \cdot f \mid a_f \in \mathbb{F}_{2^\kappa}, f \in \mathbb{F}_2[x, y]/(x^2 + x, y^2 + y) \right\}. \end{aligned}$$

For instance, using Lagrange polynomials, we can write:

$$\begin{aligned} G_{x,y} &:= G_{0,0}(x + 1)(y + 1) + G_{0,1}(x + 1)y + G_{1,0}x(y + 1) + G_{1,1}xy \in \mathbb{T}_\kappa \\ H_{x,y} &:= H_{0,0}(x + 1)(y + 1) + H_{0,1}(x + 1)y + H_{1,0}x(y + 1) + H_{1,1}xy \in \mathbb{T}_\kappa. \end{aligned} \tag{2}$$

To generate the ciphertexts, the garbler chooses $\mathbf{G} := (G_{0,0}, G_{0,1}, G_{1,0}, G_{1,1})$ so that Eq. (1) should hold for any choices of $(x, y) \in \mathbb{F}_2 \times \mathbb{F}_2$. By comparing the coefficients of 1, x , y and xy in the both sides, we obtain the following system of linear equations:

$$\begin{aligned} C + \alpha\beta\Delta &= G_{0,0} + H_{0,0} \\ \beta\Delta &= G_{0,0} + G_{1,0} + H_{0,0} + H_{1,0} \\ \alpha\Delta &= G_{0,0} + G_{0,1} + H_{0,0} + H_{0,1} \\ \Delta &= \sum_{i,j \in \mathbb{F}_2} (G_{i,j} + H_{i,j}). \end{aligned} \tag{3}$$

The garbler produces the desired ciphertexts by solving this system of equations with respect to \mathbf{G} . One might easily check that it is equivalent to the system of linear equations described in [15].

For readers, it seems to be just a matter of wordplay. Nevertheless, it provides an intriguing intuition into the understanding of garbled schemes as we shall see below.

Row Reduction Technique. In Eq. (3), all equations have the term $G_{0,0}$ in common. Therefore, simply canceling out the term will not affect on solving the system of linear equations. It allows us to set $G_{0,0} = 0 \in \mathbb{F}_{2^\kappa}$. It reduces the size of ciphertexts from 4κ to 3κ .

Half-Gate Garbling. Zahur, Rosulek and Evans [17] showed that the size of ciphertexts further reduces to 2κ . It is called the half-gate garbling technique. Their construction has two main differences: First, the ciphertexts $G_{x,y}$ are generated using only two ciphertexts G_0 and G_1 through the formula $G_{x,y} := xG_0 + yG_1$, rather than four independent ciphertexts. Second, the output labels are encrypted using the key $H(A_x) + H(B_y)$ instead of $H(A_x, B_y)$.

We provide a more general explanation of this technique from the algebraic point of view. In the technique, the garbling scheme can be represented as the following equation:

$$C + (x + \alpha)(y + \beta)\Delta = xG_0 + yG_1 + H(A_x) + H(B_y) + R_A(x, y) \cdot A_x + R_B(x, y) \cdot B_y, \tag{4}$$

where R_A, R_B are linear polynomials in $\mathbb{F}_2[x, y]$ to be determined later. Similar to Eq. (2), we write the following terms as polynomials in \mathbb{T}_κ :

$$\begin{aligned} H(A_x) &= (x + 1)H(A_0) + xH(A_1) \\ H(B_y) &= (y + 1)H(B_0) + yH(B_1) \\ A_x &= A_0 + x\Delta \\ B_y &= B_0 + y\Delta. \end{aligned} \tag{5}$$

Let us set $R_A = y$ and $R_B = 0$ which is equivalent to the setting of [17]. As before, one substitutes Eq. (5) into Eq. (4) and compares the coefficients. We have

$$\begin{aligned} C + \alpha\beta\Delta &= H(A_0) + H(B_0) \\ \beta\Delta &= G_0 + H(A_0) + H(A_1) \\ \alpha\Delta &= G_1 + H(B_0) + H(B_1). \end{aligned} \tag{6}$$

The garbler determines the output label C and the ciphertexts (G_0, G_1) using the above equation.

In general, setting $R_A = a_0 + a_1x + a_2y$ and $R_B = b_0 + b_1x + b_2y$ such that $a_2 + b_1 = 1$ leads us to a valid garbled circuit. This can be seen as follows: The expression $R_A(A_0 + x\Delta) + R_B(B_0 + y\Delta)$ contains the term $xy\Delta$ if and only if $a_2 + b_1 = 1$. Since the quadratic term $xy\Delta$ gets cancelled out in Eq. (4), the values of C and G_0, G_1 can be found by comparing the coefficients of 1, x and y .

From our algebraic viewpoint, it is interesting to observe that their modifications involving the use of $xG_0 + yG_1$ and $H(A_x) + H(B_y)$ instead of $G_{x,y}$ and $H(A_x, B_y)$ enabled the representation of the garbling equation without any quadratic terms. Consequently, this reduced the number of free variables that have to be determined.

RR21's Garbling Scheme. Rosulek and Roy [15] proposed a garbling scheme, dubbed as RR21 from now on, that further reduces the size of the ciphertexts from 2κ to $1.5\kappa + O(1)$. One of their primary ideas involves splitting the output wire label into two parts, say $C = (C^L || C^R)$, where each half is of $\kappa/2$ bits. Then they discuss how to derive each half using three well-structured $\kappa/2$ -bits ciphertexts G_0, G_1 and G_2 , along with input labels. It should be noted that a straightforward application of the half-gate strategy would not yield any

enhancements: The size of the garbled gates for each half would be of $2 \cdot (\kappa/2)$ bits.

To obtain a further reduction on the garbled gate size, the RR21 construction implicitly ensures that two of the four $\kappa/2$ -bits ciphertexts will be identical. For instance, suppose that (G_0^L, G_1^L) and (G_0^R, G_1^R) are ciphertexts for the left and right halves of C , respectively. They enforce $G_1^L = G_0^R$ and demonstrate how to construct garbling schemes satisfying this condition. To accomplish this challenging task, they propose employing extra oracle queries on $A_x + B_y$ alongside A_x and B_y . Then the idea is to use the following combination of oracle queries to mask the each half of the output labels:

$$\begin{aligned} C^L + (x + \alpha)(y + \beta)\Delta^L &= xG_0^L + yG_1^L + H(A_x) + H(A_x + B_y) + \dots, \\ C^R + (x + \alpha)(y + \beta)\Delta^R &= xG_0^R + yG_1^R + H(B_y) + H(A_x + B_y) + \dots. \end{aligned} \tag{7}$$

Keeping in mind the above, one may interpret the RR21 construction from the algebraic viewpoint. Again, we can write $H(A_x + B_y)$ as follows:²

$$H(A_x + B_y) = (x + y + 1)H(A_0 + B_0) + (x + y)H(A_0 + B_1).$$

Then we have

$$\begin{bmatrix} H(A_x) + H(A_x + B_y) \\ H(B_y) + H(A_x + B_y) \end{bmatrix} = \mathbf{M} \cdot \mathbf{H},$$

where

$$\mathbf{M} := \begin{bmatrix} x + 1 & x & 0 & 0 & x + y + 1 & x + y \\ 0 & 0 & y + 1 & y & x + y + 1 & x + y \end{bmatrix}$$

and $\mathbf{H} := (H(A_0), H(A_1), H(B_0), H(B_1), H(A_0 + B_0), H(A_0 + B_1))^\top$.

Interestingly, we observe that the y -coefficient of $H(A_x) + H(A_x + B_y)$ and the x -coefficient of $H(B_y) + H(A_x + B_y)$ are identical to each other. Let us enforce $G_1^L = G_0^R$ as desired. By properly rearranging and rewriting Eq. (7), the RR21 construction can be restated as the following equation:

$$\mathbf{V} \begin{bmatrix} \mathbf{C} \\ \mathbf{G} \end{bmatrix} = \mathbf{M}\mathbf{H} + \mathbf{R}_A(\mathbf{A}_0 + x\Delta) + \mathbf{R}_B(\mathbf{B}_0 + y\Delta) + (x + \alpha)(y + \beta)\Delta, \tag{8}$$

where

$$\mathbf{V} := \begin{bmatrix} 1 & 0 & x & 0 & x + y \\ 0 & 1 & 0 & y & x + y \end{bmatrix}, \mathbf{C} := \begin{bmatrix} C^L \\ C^R \end{bmatrix}, \mathbf{A}_0 := \begin{bmatrix} A_0^L \\ A_0^R \end{bmatrix}, \mathbf{B}_0 := \begin{bmatrix} B_0^L \\ B_0^R \end{bmatrix}, \Delta := \begin{bmatrix} \Delta^L \\ \Delta^R \end{bmatrix},$$

and $\mathbf{G} := (G_0, G_1, G_2)^\top = (G_0^L + G_0^R, G_1^L + G_1^R, G_1^L)^\top$. Here, \mathbf{R}_A and \mathbf{R}_B are 2×2 matrices over $\mathbb{F}_2[x, y]$ to be determined later.

Observe that the y -coefficient of the upper and the x -coefficient of the lower in $\mathbf{V} \begin{bmatrix} \mathbf{C} \\ \mathbf{G} \end{bmatrix}$ coincides with each other. It is exactly equivalent to saying that \mathbf{M} and

² With the free-XOR constraint, recall that $H(A_0 + B_0) = H(A_1 + B_1)$ and $H(A_0 + B_1) = H(A_1 + B_0)$.

\mathbf{V} have the same column space over \mathbb{F}_2 . Indeed, \mathbf{V} is a column-reduced matrix of \mathbf{M} where the operation is carried over \mathbb{F}_2 .

Once \mathbf{R}_A and \mathbf{R}_B are determined (as we shall describe soon how to determine them), the garbler generates the output label \mathbf{C} and the ciphertexts \mathbf{G} analogously to prior constructions, ensuring that Eq. (8) holds for all $x, y \in \mathbb{F}_2$.

Choosing Control Matrices. We now proceed to explain the procedure for determining the matrices \mathbf{R}_A and \mathbf{R}_B . According to the linear-algebraic representation in [15], this step is equivalent to find out the control matrix. Although, according to their linear-algebraic representation, they managed to identify the control matrix solely through an exhaustive computer search, we can provide explicit formulas for \mathbf{R}_A and \mathbf{R}_B due to the algebraic perspective.

First of all, we note that \mathbf{M} and \mathbf{V} share the same column space. Therefore, it suffices to guarantee that the remaining term of Eq. (8) also belongs to this common space, ensuring the existence of \mathbf{C} and \mathbf{G} satisfying Eq. (8). More precisely, it is equivalent to the following:

1. The y -coefficient of the top and the x -coefficient of the bottom in $\mathbf{R}_A(\mathbf{A}_0 + x\mathbf{\Delta}) + \mathbf{R}_B(\mathbf{B}_0 + y\mathbf{\Delta}) + (x + \alpha)(y + \beta)\mathbf{\Delta}$ coincide with each other;
2. The xy -term in $\mathbf{R}_A(\mathbf{A}_0 + x\mathbf{\Delta}) + \mathbf{R}_B(\mathbf{B}_0 + y\mathbf{\Delta}) + (x + \alpha)(y + \beta)\mathbf{\Delta}$ vanishes.

To find \mathbf{R}_A and \mathbf{R}_B satisfying the aforementioned conditions, we write $\mathbf{R}_X = \mathbf{R}_{X,0} + \mathbf{R}_{X,1}x + \mathbf{R}_{X,2}y$ for $X \in \{A, B\}$, where each $\mathbf{R}_{X,i}$ is a 2×2 binary matrix.³ Substituting this to Eq. (8) and imposing that $x^2 = x$ and $y^2 = y$ yield

$$\mathbf{V} \begin{bmatrix} \mathbf{C} \\ \mathbf{G} \end{bmatrix} = \mathbf{M}\mathbf{H} + (\mathbf{R}_{A,1}\mathbf{A}_0 + \mathbf{R}_{B,1}\mathbf{B}_0 + (\mathbf{R}_{A,0} + \mathbf{R}_{A,1} + \beta\mathbf{I})\mathbf{\Delta})x + (\mathbf{R}_{A,2}\mathbf{A}_0 + \mathbf{R}_{B,2}\mathbf{B}_0 + (\mathbf{R}_{B,0} + \mathbf{R}_{B,2} + \alpha\mathbf{I})\mathbf{\Delta})y + ((\mathbf{R}_{A,2} + \mathbf{R}_{B,1} + \mathbf{I})\mathbf{\Delta})xy + (\text{constant}), \tag{9}$$

where \mathbf{I} is the 2-dimensional identity matrix.

As the aforementioned conditions should satisfy for arbitrary choices of $\mathbf{A}_0, \mathbf{B}_0$, and $\mathbf{\Delta}$, the conditions translate to the following system of equations:

$$\begin{aligned} \mathbf{R}_{A,2} + \mathbf{R}_{B,1} + \mathbf{I}_s &= 0 \\ \begin{bmatrix} 0 & 1 \end{bmatrix} \mathbf{R}_{A,1} &= \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \mathbf{R}_{A,2} \\ \begin{bmatrix} 0 & 1 \end{bmatrix} \mathbf{R}_{B,1} &= \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \mathbf{R}_{B,2} \\ [0 \ 1] (\mathbf{R}_{A,0} + \mathbf{R}_{A,1} + \beta\mathbf{I}) &= [1 \ 0] (\mathbf{R}_{B,0} + \mathbf{R}_{B,2} + \alpha\mathbf{I}) \end{aligned} \tag{10}$$

Solving the above equations provides us the following formulas for \mathbf{R}_A and \mathbf{R}_B :

$$\begin{aligned} \mathbf{R}_{A,1} &= \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}, & \mathbf{R}_{A,2} &= \begin{bmatrix} a_3 & a_4 \\ b_3 & b_4 \end{bmatrix}, & \mathbf{R}_{A,0} &= \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} \\ \mathbf{R}_{B,1} &= \begin{bmatrix} a_3 + 1 & a_4 \\ b_3 & b_4 + 1 \end{bmatrix}, & \mathbf{R}_{B,2} &= \begin{bmatrix} b_3 & b_4 + 1 \\ e_3 & e_4 \end{bmatrix}, & \mathbf{R}_{B,0} &= \begin{bmatrix} f_1 & f_2 \\ f_3 & f_4 \end{bmatrix}, \end{aligned} \tag{11}$$

³ We observe that it is sufficient to consider the case where \mathbf{R}_X is linear rather than of arbitrary degree. This is because when \mathbf{H} consists solely of linear queries, \mathbf{M} is composed only of linear polynomials. For $\mathbf{R}_A(\mathbf{A}_0 + x\mathbf{\Delta})$ to lie within the same column space as \mathbf{M} , we deduce that $\mathbf{R}_A\mathbf{A}_0 \in \text{span}(\mathbf{M})$ for any \mathbf{A}_0 . Therefore, \mathbf{R}_A must also be linear.

where $f_1 = a_3 + b_3 + c_3 + \alpha$ and $f_2 = a_4 + b_4 + c_4 + \beta + 1$ and all the other unspecified entries are arbitrary binary elements. One might observe that $(\mathbf{R}_A, \mathbf{R}_B)$ is a 14-dimensional space.

To garble a gate, the garbler must select a pair of matrices $(\mathbf{R}_A, \mathbf{R}_B)$ from the set of 2^{14} possible choices. The garbler then needs to share this selection with the evaluator so she can utilize it in order to decrypt the encrypted gate. However, unlike the half-gate scheme, $(\mathbf{R}_A, \mathbf{R}_B)$ are dependent on the choice of the secret permute bits α and β . As a result, providing the matrices $(\mathbf{R}_A, \mathbf{R}_B)$ in clear to the evaluator would violate the privacy property of the garbling scheme. Rosulek and Roy addressed this challenge by incorporating the concept of *dicing*, initially proposed by Kempka, Kikuchi, and Suzuki in [10]. In a nutshell, the method involves encrypting the matrices $(\mathbf{R}_A, \mathbf{R}_B)$ and sending the resulting ciphertexts to the evaluator, ensuring that she can only obtain a decryption the ciphertexts on her active input labels. Precisely, if A_i and B_j are the evaluator's active input labels, then she will solely be capable of obtaining the value of $(\mathbf{R}_A(i, j), \mathbf{R}_B(i, j))$, rather than having access to the entire information on $(\mathbf{R}_A, \mathbf{R}_B)$. For further clarification, we have provided additional details regarding the dicing technique from the algebraic perspective in Appendix A.

4 Analysis on Sliced Garbling

4.1 A Generalized Framework for Garbling Schemes

In this section, we provide a generalized framework for constructing garbled gates based on the observations made in Sect. 3. For simplicity of discussion, we primarily concentrate on a gate g with three fan-ins and a single fan-out throughout this section. Nonetheless, it is evident that similar arguments are still valid for any gate g with a higher fan-in.

To begin with, we first re-establish some essential notation. Let g be a gate with input wires a, b, c and output wire d .

Notations.

- (*Permute bits*) The values $\alpha, \beta, \gamma \in \{0, 1\}$ are secret permute bits that are only known by the garbler.
- (*Input labels*) $A_\alpha, B_\beta, C_\gamma \in \{0, 1\}^\kappa$ are wire labels corresponding to the **false** value on input wires a, b and c , respectively.
- (*Output label*) $D \in \{0, 1\}^\kappa$ represents the output wire label corresponding to the **false** value on the output wire d .
- (*Color bits*) When A_x, B_y and C_z are held by the evaluator, we assume that the subscripts, x, y and z , referred to as color bits, are known by the evaluator.
- (*Free-XOR*) The value $\Delta \in \{0, 1\}^\kappa$ is a global offset secretly chosen by the garbler. Then, $W_0 + W_1 = \Delta$ for $W \in \{A, B, C\}$. Also, $D + \Delta$ represents the output label corresponding to the **true** on the wire d .

- (*Sliced labels*) Given an integer s , we represent the wire label $W = W^1 \parallel \dots \parallel W^s$ as $\mathbf{W} = (W^1, \dots, W^s)$, where each W^i is κ/s -bits. If $s = 1$, we simply write W instead of \mathbf{W} .

With the above notations, the wire labels A_x, B_y and C_z correspond to the logical values $u := x + \alpha, v := y + \beta$ and $w := z + \gamma$, where the addition is carried out over \mathbb{F}_2 .

In the following, we assume that only linear operations are allowed, apart from querying random oracles on input labels, during the construction of garbled circuits. Based on the prescribed observation, we specifically consider a garbling scheme that can be represented as an equation of the following form:

$$\mathbf{D} + g(x + \alpha, y + \beta, z + \gamma)\mathbf{\Delta} = \mathbf{W}\mathbf{G} + \mathbf{M}\mathbf{H} + \mathbf{R}_A\mathbf{A}_x + \mathbf{R}_B\mathbf{B}_y + \mathbf{R}_C\mathbf{C}_z. \tag{12}$$

Providing a concrete scheme involves the steps of specifying the matrices \mathbf{W} , \mathbf{M} , \mathbf{R}_A , \mathbf{R}_B and \mathbf{R}_C (which are over $\mathbb{F}_2[x, y, z]$) and \mathbf{H} (which are over $\mathbb{F}_{2^{\kappa/s}}$), whose detailed descriptions will be presented subsequently.

Once they are established, to garble a 3-input gate g , the garbler generates the output label \mathbf{D} and the ciphertexts \mathbf{G} according to Eq. (12). Upon receiving the ciphertexts \mathbf{G} , the evaluator computes the right-hand side of Eq. (12) using her *active* input labels A_i, B_j , and C_k for some $(i, j, k) \in \mathbb{F}_2^3$. This allows the evaluator only revealing the output label corresponding to $g(i + \alpha, j + \beta, k + \gamma)$.⁴

Now, we proceed to explain on the idea of establishing Eq. (12). It is analogous to the description we have seen from Sect. 3. It can be summarized as follows:

1. The vector \mathbf{H} defined over $\mathbb{F}_{2^{\kappa/s}}$ consists of all possible responses of oracle queries that can be made during the construction of garbled circuits.
2. The matrix \mathbf{M} is a matrix over $\mathbb{F}_2[x, y, z]$ with s rows. Each row of \mathbf{M} is determined by which combinations of oracle queries are to be used for the corresponding slices.
3. The matrix $\mathbf{V} := [\mathbf{I}|\mathbf{W}]$ is chosen so that it has the same column space (over \mathbb{F}_2) with the matrix \mathbf{M} . Here, the matrix \mathbf{I} is the s -dimensional identity matrix. Note that we implicitly assumed that the column-reduced matrix of \mathbf{M} contains the identity matrix.
4. Each of the matrices \mathbf{R}_X for $X \in \{A, B, C\}$ is chosen so that the vector $\mathbf{R}_A\mathbf{A}_x + \mathbf{R}_B\mathbf{B}_y + \mathbf{R}_C\mathbf{C}_z + g(x + \alpha, y + \beta, z + \gamma)\mathbf{\Delta}$ belongs to the same space spanned by the columns of \mathbf{V} .

From the preceding discussion, we observe that primary concerns in garbling constructions revolve around (1) determining the matrix \mathbf{M} and the vector \mathbf{H} , and (2) deriving the matrices \mathbf{R}_X for each $X \in \{A, B, C\}$. Following the previous works, we refer the matrix \mathbf{R}_X to the *control matrix*.

⁴ To simplify discussion, for now, let us presume that the evaluator obtains the values of $\mathbf{R}_A, \mathbf{R}_B$ and \mathbf{R}_C precisely at (i, j, k) in a certain way. In other words, we implicitly assume the dicing technique is applied.

4.2 Garbled Circuits for 3-Input Gates

Recently, Ashur, Hazay and Satish [1] proposed a scheme that garbles a 3-input gate. They specifically claimed that a circuit of the form $g_{tri}(u, v, w) := u(v + w)$ can be garbled at a cost of $4/3\kappa + O(1)$ bits. As its corollary, they insisted that garbling an AND gate requires the same cost as garbling g_{tri} by fixing $w = 0$. This claim, if correct, would provide an asymptotic improvement compared to the state-of-the-art requiring $3/2\kappa + O(1)$ bits.

However, in this paper, we demonstrate that their proposed construction will leak permute bits, thereby jeopardizing the security guarantees offered by garbling schemes. Prior to describe our main results, we begin by reviewing their construction from our perspective provided in Sect. 4.1.

4.2.1 Review on Tri-Gate Garbling Scheme

Choice of the Matrix \mathbf{M} and the Vector \mathbf{H} . In their work [1], they suggested splitting wire labels into three slices, i.e. $s = 3$, to garble the gate g_{tri} . To encrypt each slices of the output label $\mathbf{D} = (D^1, D^2, D^3)$, they chose the following linear combinations of oracle queries:

$$\begin{aligned} D^1 + g_{tri}(u, v, w)\Delta^1 &= H(A_x) + H(B_y) + H(A_x + B_y + C_z) + \dots \\ D^2 + g_{tri}(u, v, w)\Delta^2 &= H(B_y) + H(C_z) + H(A_x + B_y + C_z) + \dots \\ D^3 + g_{tri}(u, v, w)\Delta^3 &= H(A_x) + H(C_z) + H(A_x + B_y + C_z) + \dots, \end{aligned}$$

where $u := x + \alpha, v := y + \beta$ and $w := z + \gamma$.

Analogously to the previous constructions, we represent them into polynomials as follows. For instance, under the free-XOR setting, we may write

$$H(A_x + B_y + C_z) = (x + y + z + 1)H(A_0 + B_0 + C_0) + (x + y + z)H(A_0 + B_0 + C_1).$$

Let us define

$$\mathbf{H} := \begin{bmatrix} H(A_0) \\ H(A_1) \\ H(B_0) \\ H(B_1) \\ H(C_0) \\ H(C_1) \\ H(A_0 + B_0 + C_0) \\ H(A_0 + B_0 + C_1) \end{bmatrix}.$$

Then this determines the matrix \mathbf{M} in Eq. (12) as follows:

$$\mathbf{M} = \begin{bmatrix} x + 1 & x & y + 1 & y & 0 & 0 & x + y + z + 1 & x + y + z \\ 0 & 0 & y + 1 & y & z + 1 & z & x + y + z + 1 & x + y + z \\ x + 1 & x & 0 & 0 & z + 1 & z & x + y + z + 1 & x + y + z \end{bmatrix}.$$

It can be easily verified that the following matrix \mathbf{V} has the same column space as \mathbf{M} , where the span is carried out over $\mathbb{F}_{2^{\kappa/3}}$:

$$\mathbf{V} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & x & y & 0 & x+y+z \\ 0 & 1 & 0 & 0 & y & z & x+y+z \\ 0 & 0 & 1 & x & 0 & z & x+y+z \end{array} \right] := [\mathbf{I} \mid \mathbf{W}]. \tag{13}$$

Here, \mathbf{I} is the 3-dimensional identity matrix and \mathbf{W} is the right (3×4) -matrix.

Choosing the Control Matrix \mathbf{R}_X . Having determined the matrix \mathbf{M} , the next step is to choose the control matrix \mathbf{R}_X . Based on the linear-algebraic representation by Rosulek and Roy [15], Ashur, Hazay and Satish [1] demonstrated how to find the control matrices. While they elucidated the methodology for finding these matrices, explicit formulas were not provided.

In the following, we provide explicit formulas for the control matrices based on our algebraic perspective. As desired, this will show that how their proposed construction leaks the secret permuted bits, even though the dicing technique is properly applied.

We begin with scrutinizing the subspace spanned by the columns of $\mathbf{V} = [\mathbf{I} \mid \mathbf{W}]$. Let us consider

$$span(\mathbf{V}) := \left\{ \mathbf{V} \cdot \begin{bmatrix} \mathbf{D} \\ \mathbf{G} \end{bmatrix} \mid \mathbf{D} \in \mathbb{F}_{2^{\kappa/3}}^3, \mathbf{G} \in \mathbb{F}_{2^{\kappa/3}}^4 \right\}.$$

We represent the matrix \mathbf{W} as follows:

$$\mathbf{W} = \mathbf{W}_1x + \mathbf{W}_2y + \mathbf{W}_3z,$$

where each \mathbf{W}_i is a (3×4) -binary matrix derived from \mathbf{W} . Note that these matrices are formed by considering the coefficients of x , y , and z in the expansion of \mathbf{W} .

Let us consider a cokernel matrix $\mathbf{P} = [\mathbf{P}_1 \mid \mathbf{P}_2 \mid \mathbf{P}_3]$ of a binary matrix formed by $\begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_2 \\ \mathbf{W}_3 \end{bmatrix}$. In other words, we have the following:

$$\mathbf{P}_1\mathbf{W}_1 + \mathbf{P}_2\mathbf{W}_2 + \mathbf{P}_3\mathbf{W}_3 = 0,$$

where each \mathbf{P}_i is a (5×3) binary matrix.

Given $\boldsymbol{\nu} \in span(\mathbf{V})$, we represent the vector $\boldsymbol{\nu}$ as $\boldsymbol{\nu} = \boldsymbol{\nu}_0 + \boldsymbol{\nu}_1x + \boldsymbol{\nu}_2y + \boldsymbol{\nu}_3z$, where $\boldsymbol{\nu}_i = \mathbf{W}_i\mathbf{G}$ for some $\mathbf{G} \in \mathbb{F}_{2^{\kappa/3}}^4$. Thus we have that

$$\mathbf{P}_1\boldsymbol{\nu}_1 + \mathbf{P}_2\boldsymbol{\nu}_2 + \mathbf{P}_3\boldsymbol{\nu}_3 = 0 \text{ for all } \boldsymbol{\nu} \in span(\mathbf{V}).$$

We shall use this relation to find the control matrices \mathbf{R}_X . We define this relation, denoted by $\pi_{\mathbf{V}}$, as:

$$\pi_{\mathbf{V}}(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3) := \mathbf{P}_1\mathbf{X}_1 + \mathbf{P}_2\mathbf{X}_2 + \mathbf{P}_3\mathbf{X}_3,$$

where each \mathbf{X}_i refers to a binary matrix with the same number of rows as the columns in \mathbf{P}_i .

In summary, the span by the columns of \mathbf{V} is given by

$$\text{span}(\mathbf{V}) = \{\boldsymbol{\nu} \mid \boldsymbol{\nu} = \boldsymbol{\nu}_0 + \boldsymbol{\nu}_1x + \boldsymbol{\nu}_2y + \boldsymbol{\nu}_3z \text{ and } \pi_{\mathbf{V}}(\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \boldsymbol{\nu}_3) = 0 \text{ for } \boldsymbol{\nu}_i \in \mathbb{F}_{2^{\kappa/3}}^3\}.$$

Recall that each \mathbf{R}_X has to be chosen so that

$$\begin{aligned} \rho &:= \mathbf{R}_A\mathbf{A}_x + \mathbf{R}_B\mathbf{B}_y + \mathbf{R}_C\mathbf{C}_z + g_{tri}(x + \alpha, y + \beta, z + \gamma)\boldsymbol{\Delta} \\ &= \mathbf{R}_A\mathbf{A}_0 + \mathbf{R}_B\mathbf{B}_0 + \mathbf{R}_C\mathbf{C}_0 \\ &\quad + (x\mathbf{R}_A + y\mathbf{R}_B + z\mathbf{R}_C + (x + \alpha)(y + z + \beta + \gamma)\mathbf{I})\boldsymbol{\Delta} \in \text{span}(\mathbf{V}), \end{aligned} \quad (14)$$

for any choices of $\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0$, and $\boldsymbol{\Delta}$. Since $\rho := \rho_0 + \rho_1x + \rho_2y + \rho_3z \in \text{span}(\mathbf{V})$, we also have $\pi_{\mathbf{V}}(\rho_1, \rho_2, \rho_3) = 0$. This condition must be satisfied for arbitrary choices of $\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0$, and $\boldsymbol{\Delta}$. Consequently, it is sufficient to analyze the equivalent relationships by focusing on the coefficients of $\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0$, and $\boldsymbol{\Delta}$ in Eq. (14) independently.

For instance, in Eq. (14), the vector $\mathbf{R}_A\mathbf{A}_0$ should belong to $\text{span}(\mathbf{V})$ for any \mathbf{A}_0 . Let us express $\mathbf{R}_A = \mathbf{R}_{A,0} + \mathbf{R}_{A,1}x + \mathbf{R}_{A,2}y + \mathbf{R}_{A,3}z$, where each $\mathbf{R}_{A,i}$ is a 3×3 binary matrix. Then, the requirement that $\mathbf{R}_A\mathbf{A}_0 \in \text{span}(\mathbf{V})$ for all \mathbf{A}_0 is equivalent to the following condition:

$$\pi_{\mathbf{V}}(\mathbf{R}_{A,1}, \mathbf{R}_{A,2}, \mathbf{R}_{A,3}) = 0.$$

Similar reasoning applies to the matrices \mathbf{R}_B and \mathbf{R}_C as well. As a result, we have

$$\pi_{\mathbf{V}}(\mathbf{R}_{X,1}, \mathbf{R}_{X,2}, \mathbf{R}_{X,3}) = 0 \text{ for each } X \in \{A, B, C\}. \quad (15)$$

As we are interested in the case that x, y and z take the values in \mathbb{F}_2 , we impose the condition that $x^2 = x, y^2 = y$, and $z^2 = z$. Then, the $\boldsymbol{\Delta}$ -term in Eq. (14) can be rewritten as follows:

$$\begin{aligned} \rho_{\Delta} &:= x\mathbf{R}_A + y\mathbf{R}_B + z\mathbf{R}_C + (x + \alpha)(y + z + \beta + \gamma)\mathbf{I} \\ &= (\mathbf{R}_{A,0} + \mathbf{R}_{A,1} + (\beta + \gamma)\mathbf{I})x + (\mathbf{R}_{B,0} + \mathbf{R}_{B,2} + \alpha\mathbf{I})y + (\mathbf{R}_{C,0} + \mathbf{R}_{C,3} + \alpha\mathbf{I})z + \\ &\quad (\mathbf{R}_{A,2} + \mathbf{R}_{B,1} + \mathbf{I})xy + (\mathbf{R}_{A,3} + \mathbf{R}_{C,1} + \mathbf{I})xz + (\mathbf{R}_{B,3} + \mathbf{R}_{C,2})yz. \end{aligned}$$

The requirement that $\rho_{\Delta}\boldsymbol{\Delta} \in \text{span}(\mathbf{V})$ for all $\boldsymbol{\Delta}$ is equivalent to the following equations:

$$\begin{aligned} \pi_{\mathbf{V}}(\mathbf{R}_{A,0} + \mathbf{R}_{A,1} + (\beta + \gamma)\mathbf{I}, \mathbf{R}_{B,0} + \mathbf{R}_{B,2} + \alpha\mathbf{I}, \mathbf{R}_{C,0} + \mathbf{R}_{C,3} + \alpha\mathbf{I}) &= 0 \\ \mathbf{R}_{A,2} + \mathbf{R}_{B,1} + \mathbf{I} &= 0 \\ \mathbf{R}_{A,3} + \mathbf{R}_{C,1} + \mathbf{I} &= 0 \\ \mathbf{R}_{B,3} + \mathbf{R}_{C,2} &= 0 \end{aligned} \quad (16)$$

We can construct binary matrices $\mathbf{R}_{X,i}$ such that they fulfill the requirements imposed by Eq. (15) and (16). Then it will provide explicit formulas for the set of all possible pairs of the control matrices. Although proving how this construction exposes the permutation bits does not require comprehensive explicit formulae, we nonetheless supply more details regarding the identification of $\mathbf{R}_{X,i}$ and their resulting explicit formulae in Appendix B for the sake of thoroughness.

4.2.2 Analysis on Tri-Gate Garbling

In the rest of this section, we show that the garbling scheme suggested by [1] is insecure. To be precise, we show that the evaluator can reveal the values of α and $\beta + \gamma$. Consequently, she would be aware of the logical values associated with the input labels \mathbf{A}_0 and $\mathbf{B}_0 + \mathbf{C}_0$. Furthermore, if the evaluator has prior information about either β or γ , she can deduce all logical values related to her active input labels.

As we have explicit formulae for the control matrices $(\mathbf{R}_A, \mathbf{R}_B, \mathbf{R}_C)$, as shown in Eq. (22), it becomes straightforward to verify the aforementioned claims. For instance, if the evaluator’s active input labels are $\mathbf{A}_0, \mathbf{B}_0$ and \mathbf{C}_0 , then she will have the values of $(\mathbf{R}_A, \mathbf{R}_B, \mathbf{R}_C)$ evaluated at $(0, 0, 0)$. In other words, the evaluator is aware of the matrix values $(\mathbf{R}_{A,0}, \mathbf{R}_{B,0}, \mathbf{R}_{C,0})$. As one might observe from Eq. (22), this directly implies the desired results: By adding the second row of $\mathbf{R}_{A,0}$ to its third row, the evaluator obtains the vector $(\beta + \gamma, 0, \beta + \gamma)$. Likewise, if the evaluator adds the first row of $\mathbf{R}_{B,0}$ with its second row, she will obtain the vector $(\alpha, \alpha, 0)$.

In the following, we also provide alternative mathematical arguments for the above claims without requiring explicit formulae for the control matrices. We begin with the following simple observations:

1. Assuming that $\pi_{\mathbf{V}}(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3) = 0$ and $\pi_{\mathbf{V}}(\mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3) = 0$, it follows that

$$\pi_{\mathbf{V}}(\mathbf{X}_1 + \mathbf{Y}_1, \mathbf{X}_2 + \mathbf{Y}_2, \mathbf{X}_3 + \mathbf{Y}_3) = 0.$$

Each of \mathbf{X}_i and \mathbf{Y}_i are binary matrices with identical dimensions and an equal number of rows as the columns in \mathbf{P}_i .

2. For the matrix $\mathbf{P} = [\mathbf{P}_1 \mid \mathbf{P}_2 \mid \mathbf{P}_3]$, as given in Eq. (21), there exists non-zero vectors $\mathbf{k}_1, \mathbf{k}_2$ and \mathbf{k}_3 such that

$$\mathbf{k}_i^\top \mathbf{P}_{i-1} = \mathbf{k}_i^\top \mathbf{P}_{i+1} = 0 \text{ and } \mathbf{k}_i^\top \mathbf{P}_i \neq 0$$

for every $i = 1, 2, 3$. Here, the subscript indices are computed modulo 3. For instance, one might choose $\mathbf{k}_1^\top = (1, 0, 0, 0, 0)$, $\mathbf{k}_2^\top = (0, 1, 0, 0, 0)$, and $\mathbf{k}_3^\top = (0, 0, 1, 0, 0)$.

Based on the above observations, we prove the following theorem:

Theorem 1. *Let \mathbf{V} be as Eq. (13). For a fixed triple $(\alpha, \beta, \gamma) \in \mathbb{F}_2^3$, consider a set \mathcal{R} of three matrices $(\mathbf{R}_A, \mathbf{R}_B, \mathbf{R}_C)$ such that the vector ρ defined in Eq. (14) belongs to $\text{span}(\mathbf{V})$ for any choices of $\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0$, and Δ . Take an element $(\mathbf{R}_A, \mathbf{R}_B, \mathbf{R}_C)$ from \mathcal{R} . As before, let us write $\mathbf{R}_X = \mathbf{R}_{X,0} + \mathbf{R}_{X,1}x + \mathbf{R}_{X,2}y + \mathbf{R}_{X,3}z$. Given the triple of matrices $(\mathbf{R}_A(i, j, k), \mathbf{R}_B(i, j, k), \mathbf{R}_C(i, j, k))$ for $i, j, k \in \{0, 1\}$, one can compute the values of α and $\beta + \gamma$.*

Proof. First, let us recall that $\rho \in \text{span}(\mathbf{V})$ implies that Eq. (15) and (16). We prove the assertion case by case.

Case 1. $(i, j, k) = (0, 0, 0)$: Recall that we are working over a field of characteristic 2. By adding the first equation in Eq. (16) with $\pi_{\mathbf{V}}(\mathbf{R}_{A,1}, \mathbf{R}_{A,2}, \mathbf{R}_{A,3}) = 0$ (see Eq. (15)), we obtain the following:

$$\pi_{\mathbf{V}}(\mathbf{R}_{A,0} + (\beta + \gamma)\mathbf{I}, \mathbf{R}_{B,0} + \mathbf{R}_{B,2} + \mathbf{R}_{A,2} + \alpha\mathbf{I}, \mathbf{R}_{C,0} + \mathbf{R}_{C,3} + \mathbf{R}_{A,3} + \alpha\mathbf{I}) = 0. \quad (17)$$

Multiplying \mathbf{k}_1^\top to both sides of Eq. (17), we obtain:

$$\mathbf{k}_1^\top \mathbf{P}_1 (\mathbf{R}_{A,0} + (\beta + \gamma)\mathbf{I}) = 0 \implies \mathbf{k}_1^\top \mathbf{P}_1 \mathbf{R}_{A,0} = (\beta + \gamma)\mathbf{k}_1^\top \mathbf{P}_1.$$

Since $\mathbf{R}_A(0, 0, 0) = \mathbf{R}_{A,0}$ is given, calculating $\mathbf{k}_1^\top \mathbf{P}_1 \mathbf{R}_{A,0}$ and examining whether it equals zero or not allows one to determine the bit of $\beta + \gamma$.

Similarly, adding the first equation in Eq. (16) with $\pi_{\mathbf{V}}(\mathbf{R}_{A,1}, \mathbf{R}_{A,2}, \mathbf{R}_{A,3}) = 0$ leads us to obtain the equation of the following form:

$$\pi_{\mathbf{V}}(*, \mathbf{R}_{B,0} + \alpha\mathbf{I}, *) = 0.$$

Multiplying \mathbf{k}_2^\top to the both side of the equation, we obtain:

$$\mathbf{k}_2^\top \mathbf{P}_2 (\mathbf{R}_{B,0} + \alpha\mathbf{I}) = 0 \implies \mathbf{k}_2^\top \mathbf{P}_2 \mathbf{R}_{B,0} = \alpha\mathbf{k}_1^\top \mathbf{P}_1.$$

From this relation, one can deduce the bit of α .

Case 2. $(i, j, k) = (1, 0, 0)$: In this case $\mathbf{R}_A(1, 0, 0) = \mathbf{R}_{A,0} + \mathbf{R}_{A,1}$ is given. Multiplying \mathbf{k}_1^\top to the first equation of Eq. (16) provides us

$$\mathbf{k}_1^\top \mathbf{P}_1 (\mathbf{R}_{A,0} + \mathbf{R}_{A,1} + (\beta + \gamma)\mathbf{I}) = 0.$$

Thus, one can obtain the value of $\beta + \gamma$ by calculating $\mathbf{k}_1^\top \mathbf{P}_1 (\mathbf{R}_{A,0} + \mathbf{R}_{A,1})$.

To deduce the bit of α , we observe the followings: By adding $\pi_{\mathbf{V}}(\mathbf{R}_{A,1}, \mathbf{R}_{A,2}, \mathbf{R}_{A,3}) = 0$ and $\pi_{\mathbf{V}}(\mathbf{R}_{B,1}, \mathbf{R}_{B,2}, \mathbf{R}_{B,3}) = 0$ to the first equation of Eq. (16), we obtain the equation of the form

$$\pi_{\mathbf{V}}(*, \mathbf{R}_{B,0} + \mathbf{R}_{A,2} + \alpha\mathbf{I}, *) = 0.$$

Since $\mathbf{R}_{A,2} = \mathbf{R}_{B,1} + \mathbf{I}$, we have $\pi_{\mathbf{V}}(*, \mathbf{R}_{B,0} + \mathbf{R}_{B,1} + (\alpha + 1)\mathbf{I}, *) = 0$. Therefore, calculating the value of $\mathbf{k}_2^\top \mathbf{R}_B(1, 0, 0)$ provides us the value of α .

Case 3. $(i, j, k) = (0, 1, 0)$: For the value of α , we can just use the first relation in Eq. (16). To obtain $\beta + \gamma$, use Eq. (15) with $X = A$ and B and $\mathbf{R}_{B,1} = \mathbf{R}_{A,2} + \mathbf{I}$.

Case 4. $(i, j, k) = (0, 0, 1)$: For the value of α , it is sufficient with the first relation in Eq. (16). To obtain $\beta + \gamma$, use Eq. (15) with $X = A$ and C and $\mathbf{R}_{C,1} = \mathbf{R}_{A,3} + \mathbf{I}$.

For the other cases, we can proceed analogous arguments to obtain the desired results. We leave verifying them to readers. \square

5 (Im)possibility of Higher Fan-In Gates Garbling

In this section, we build upon our earlier work and investigate the existence of secure garbling schemes for ℓ -input gates using s -sliced labels, where $\ell \geq 3$ and $s \geq 3$. Our previous research demonstrated that the construction presented in [1], which aims to garble tri-gates g_{tri} , leaks information about the permutation bits, rendering the scheme insecure. This construction relied on employing three-sliced labels to garble 3-input gates. The primary goal of this section is to address the following open question: Is there a secure garbling scheme for ℓ -input gates utilizing s -sliced labels, with $\ell \geq 3$ and $s \geq 3$? Unfortunately, our response is negative. Even considering $s \geq 3$ and $\ell \geq 3$ seems unlikely to yield a secure garbling scheme.

Let us begin with defining several notions. Throughout this section, we consider garbling an ℓ -input gate g with input wires a_1, \dots, a_ℓ and output wire b .

Notations.

- (*Permute bits*) The values $\alpha_j \in \{0, 1\}$ for $j \in \{1, \dots, \ell\}$ are secret permute bits that are only known by the garbler.
- (*Input labels*) $A_{\alpha_j, j} \in \{0, 1\}^\kappa$ are wire labels corresponding to the **false** value on the input wire a_j for each $j \in \{1, \dots, \ell\}$.
- (*Output label*) $B \in \{0, 1\}^\kappa$ represents the output wire label corresponding to the **false** value on the output wire b .
- (*Color bits*) When $A_{x_j, j}$ are held by the evaluator, we assume that the subscripts x_j , referred to as color bits, are known by the evaluator.
- (*Free-XOR*) The value $\Delta \in \{0, 1\}^\kappa$ is a global offset secretly chosen by the garbler. Then, $A_{0, j} + A_{1, j} = \Delta$ for each $j \in \{1, \dots, \ell\}$. Also, $B + \Delta$ represents the output label corresponding to the **true** on the wire b .
- (*Sliced labels*) Given an integer s , we represent the wire label $W = W^1 \parallel \dots \parallel W^s$ as $\mathbf{W} = (W^1, \dots, W^s)$, where each W^i is κ/s -bits.

Using the previously mentioned notation, we now define several matrices and vectors to establish a *garbling equation*.

- (*Linear queries*) We assume that queries to a random oracle are made on functions of the input labels. For each $j \in \{1, \dots, \ell\}$, only one of $\mathbf{A}_{0, j}$ or $\mathbf{A}_{1, j}$ can be used as inputs to the random oracle. Particularly, we restrict our concern to the setting where the only possible queries allowed to the oracle are linear functions of the input labels. That is, we consider queries of the following form: $H(\sum_{j \in I} \mathbf{A}_{x_j, j})$ for a subset $I \subset \{1, \dots, \ell\}$. We call such queries as *linear queries*.
- (*Function representation of queries*) We continue to focus on the free-XOR setting. Given an oracle response of the form $H(\sum_{j \in I} \mathbf{A}_{x_j, j})$, we represent it as a function in $\mathbb{F}_{2^{\kappa/s}}[x_1, \dots, x_\ell]$:

$$\begin{aligned}
 H(\sum_{j \in I} \mathbf{A}_{x_j, j}) &= (1 + \sum_{j \in I} x_j)H\left(\sum_{j \in I} \mathbf{A}_{0, j}\right) \\
 &\quad + (\sum_{j \in I} x_j)H\left(\mathbf{A}_{1, i_*} + \sum_{j \in I \setminus \{i_*\}} \mathbf{A}_{0, j}\right),
 \end{aligned}$$

where i_* is the index such that $x_{i_*} = 1$. To understand why this representation holds, note that if there are even number of $j \in I$ such that $x_j = 1$, then we have $H(\sum_{j \in I} \mathbf{A}_{x_j,j}) = H(\sum_{j \in I} \mathbf{A}_{0,j})$ due to the free-XOR condition. Similar arguments apply for the odd case.

- (*Query matrix*) Consider a vector \mathbf{H} comprising $H(\sum_{j \in I} \mathbf{A}_{0,j})$ and $H(\mathbf{A}_{1,i_*} + \sum_{j \in I \setminus \{i_*\}} \mathbf{A}_{0,j})$ for all non-empty subset $I \subset \{1, \dots, \ell\}$. Based upon the aforementioned function representation of $h_I := H(\sum_{j \in I} \mathbf{A}_{x_j,j})$, one can represent a linear sum of the form $\sum_I h_I$ as a dot product $\mathbf{M}^\top \cdot \mathbf{H}$, where \mathbf{M} comprises polynomials $1 + \sum_{j \in I} x_j$ and $\sum_{j \in I} x_j$ for certain I 's. In this manner, for s linear sums of $\sum_I h_I$'s, one can express $(h_1, \dots, h_s)^\top = \mathbf{M}\mathbf{H}$, where the k -th row of \mathbf{M} corresponds to the vector \mathbf{M} associated with h_j . We refer to the matrix \mathbf{M} as a *query matrix* and the vector \mathbf{H} as a *hash vector*.
- (*Control matrix*) For each $j \in \{1, \dots, \ell\}$, we introduce a j -th *control matrix*, denoted by \mathbf{R}_j , associated with the j -th input. The control matrix \mathbf{R}_j is a $(s \times s)$ -matrix with its entries in $\mathbb{F}_2[\alpha_1, \dots, \alpha_\ell, x_1, \dots, x_\ell]$. Since our main interest lies in the case when $\alpha_i \in \mathbb{F}_2$, to streamline notation without causing confusion, we treat its entries as elements in $\mathbb{F}_2[x_1, \dots, x_\ell]$, and their coefficients are parameterised by α_i 's. As before, we also write

$$\mathbf{R}_j = \mathbf{R}_{j,0} + \mathbf{R}_{j,1}x_1 + \dots + \mathbf{R}_{j,\ell}x_\ell,$$

where each $\mathbf{R}_{j,k}$ is a binary matrix.⁵

Remark 1. Assume that arbitrary *non-linear* functions of the input labels are used to make queries to the random oracle. We observe that it remains feasible to provide a function representation of the responses to these oracle queries. For instance, as we have seen from an example of Yao's garbling scheme, a response of the form $H(A_x, B_y)$ can be represented as a quadratic function. Nevertheless, we confine our attention to the setting of linear queries, consistent with prior studies [1, 15, 17]. Introducing non-linear queries may potentially increase communication costs since we must account for additional monomials resulting from higher-degree functions being considered. Notably, if our aim is to garble a degree-2 gate, it is unnecessary to consider non-linear queries of degree exceeding 2. Nonetheless, exploring the potential benefits of such non-linear queries in reducing communication cost when targeting higher-degree gates constitutes an intriguing subject for future investigation. We reserve this issue for further research. □

5.1 Garbling Equations

We are ready to formally define the notion of garbling equations. We will continue to utilize the previously introduced notation throughout this discussion.

⁵ By a similar argument to (See Footnote 3), it suffices to consider only linear polynomials.

Definition 1 (Garbling Equation). For each $i \in \{1, \dots, s\}$, let h_i be a linear sum of linear queries. For $j \in \{1, \dots, \ell\}$, let \mathbf{R}_j be a j -th control matrix. The matrix \mathbf{M} is the query matrix and the vector \mathbf{H} is the hash vector associated with (h_1, \dots, h_s) such that $(h_1, \dots, h_s) = \mathbf{M}\mathbf{H}$. Given the vector (h_1, \dots, h_s) and the control matrix \mathbf{R}_j , we define a garbling equation \mathcal{G} of a ℓ -input gate g by the following equation:

$$\mathcal{G}_g : \mathbf{V} \begin{bmatrix} \mathbf{C} \\ \mathbf{G} \end{bmatrix} = \mathbf{M}\mathbf{H} + \sum_{1 \leq j \leq \ell} \mathbf{R}_j \mathbf{A}_{x_j, j} + g(x_1 + \alpha_1, \dots, x_\ell + \alpha_\ell) \mathbf{\Delta}.$$

Here, \mathbf{V} is a matrix comprised of column basis of the \mathbb{F}_2 -subspace generated by the columns of \mathbf{M} and $[\mathbf{C}, \mathbf{G}]$ is a $(s + r)$ -dimensional vector where $s + r$ is the column length of \mathbf{V} .

Definition 2. For an ℓ -input gate g , let $\Pi = (\text{Gb}, \text{En}, \text{Ev}, \text{De})$ be a garbling scheme associated with a garbling equation \mathcal{G}_g . If the garbling scheme Π is correct, then we say that Π is **correctly garbleable** with respect to the garbling equation \mathcal{G}_g . Moreover, if Π satisfies privacy property, then we call that Π is **privately garbleable**.

We observe that if there exists a vector $[\mathbf{C}, \mathbf{G}]$ satisfying the garbling equation \mathcal{G}_g holds for any input labels $\mathbf{A}_{x_j, j}$, a global offset $\mathbf{\Delta}$, and permute bits $\alpha_j \in \mathbb{F}_2$, then the corresponding garbling scheme Π is correct.

In what follows, we will expand upon the discussions presented earlier to encompass the garbling of arbitrary ℓ -input gates.

Definition 3. Assume that \mathbf{V} is of the form $\mathbf{V} = [\mathbf{I}_s \mid \mathbf{W}]$, where \mathbf{I}_s is the s -dimensional identity matrix and \mathbf{W} is a $(s \times r)$ -matrix over $\mathbb{F}_2[x_1, \dots, x_\ell]$. As before, we write $\mathbf{W} = \mathbf{W}_1 x_1 + \dots + \mathbf{W}_\ell x_\ell$ for a $(s \times r)$ -binary matrix. Let us consider a cokernel matrix \mathbf{P} of the $(\ell s \times r)$ -matrix formed by $[\mathbf{W}_1^\top \mid \dots \mid \mathbf{W}_\ell^\top]^\top$. Abusing the notation, we denote the matrix \mathbf{P} by $\text{coker}(\mathbf{W})$, i.e. $\text{coker}(\mathbf{W})$ is a binary matrix of the following form:

$$\text{coker}(\mathbf{W}) := \mathbf{P} = \text{coker} \left(\begin{bmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_\ell \end{bmatrix} \right).$$

Moreover, if the matrix $[\mathbf{W}_1^\top \mid \dots \mid \mathbf{W}_\ell^\top]^\top$ is of rank r , then we can write $\text{coker}(\mathbf{W}) = \mathbf{P} = [\mathbf{P}_1 \mid \dots \mid \mathbf{P}_\ell]$, where each \mathbf{P}_j is a $(\ell s - r) \times s$ dimensional matrix.

For a finite field \mathbb{F} of characteristic 2, we consider the \mathbb{F} -subspace spanned by the columns in \mathbf{V} and denote it by $\text{span}(\mathbf{V})$:⁶

$$\text{span}(\mathbf{V}) = \left\{ \mathbf{v} \begin{bmatrix} \mathbf{C} \\ \mathbf{G} \end{bmatrix} \mid \mathbf{C} \in \mathbb{F}^s, \mathbf{G} \in \mathbb{F}^r \right\}.$$

⁶ In the case that s -sliced labels used, we typically choose $\mathbb{F} = \mathbb{F}_{2^{r/s}}$.

Similar to the previous sections, let us write $\boldsymbol{\nu} \in \text{span}(\mathbf{V})$ as $\boldsymbol{\nu} = \boldsymbol{\nu}_0 + \boldsymbol{\nu}_1 x_1 + \cdots + \boldsymbol{\nu}_\ell x_\ell$ for a \mathbb{F} -vector $\boldsymbol{\nu}_j$. Then we can check that

$$\boldsymbol{\nu} \in \text{span}(\mathbf{V}) \text{ if and only if } \mathbf{P}_1 \boldsymbol{\nu}_1 + \cdots + \mathbf{P}_\ell \boldsymbol{\nu}_\ell = 0.$$

Again, we define the relation $\pi_{\mathbf{V}}$ by

$$\pi_{\mathbf{V}}(\mathbf{X}_1, \dots, \mathbf{X}_\ell) := \mathbf{P}_1 \mathbf{X}_1 + \cdots + \mathbf{P}_\ell \mathbf{X}_\ell,$$

where the dimension of each matrix \mathbf{X}_j is properly defined.

5.2 Main Results

Next, we discuss on the conditions under which the garbling scheme Π is correct.

Lemma 1. *Assume that $\text{span}(\mathbf{M}) = \text{span}(\mathbf{V})$ in the garbling equation \mathcal{G}_g . If $\rho := \sum_{1 \leq j \leq \ell} \mathbf{R}_j \mathbf{A}_{x_j, j} + g(x_1 + \alpha_1, \dots, x_\ell + \alpha_\ell) \boldsymbol{\Delta} \in \text{span}(\mathbf{V})$ for any $\mathbf{A}_{x_j, j}$ and $\boldsymbol{\Delta}$, then the garbling scheme Π is correct.*

Proof. It is obvious from the definition. Since $\text{span}(\mathbf{M}) = \text{span}(\mathbf{V})$, there exist \mathbf{C}_M and \mathbf{G}_M such that $\mathbf{M}\mathbf{H} = \mathbf{V}[\mathbf{C}_M^\top \mid \mathbf{G}_M^\top]^\top$. Similarly, as we have $\rho \in \text{span}(\mathbf{V})$, there exist \mathbf{C}_ρ and \mathbf{G}_ρ such that $\rho = \mathbf{V}[\mathbf{C}_\rho^\top \mid \mathbf{G}_\rho^\top]^\top$. Therefore, we have $\mathbf{C} = \mathbf{C}_M + \mathbf{C}_\rho$ and $\mathbf{G} = \mathbf{G}_M + \mathbf{G}_\rho$. \square

In the following, let us write the ℓ -variate gate g as a polynomial of the following form:

$$g(x_1 + \alpha_1, \dots, x_\ell + \alpha_\ell) = g^{(0)}(\alpha_1, \dots, \alpha_\ell) + \sum_{d \geq 1} g_{i_1, \dots, i_d}^{(d)}(\alpha_1, \dots, \alpha_\ell) x_{i_1} \cdots x_{i_d}, \quad (18)$$

where each $g_{i_1, \dots, i_d}^{(d)}$ is the coefficient of $x_{i_1} \cdots x_{i_d}$ in the expansion of g .

Lemma 2. *Let the notations as above. Assume that \mathbf{H} only consists of linear queries. Then, we have $\rho \in \text{span}(\mathbf{V})$ for any $\mathbf{A}_{x_j, j}$ and $\boldsymbol{\Delta}$ if and only if (1) The degree of g is less than or equal to 2 and (2) the following equations hold:*

$$\begin{aligned} \pi_{\mathbf{V}}(\mathbf{R}_{j,1}, \dots, \mathbf{R}_{j,\ell}) &= 0 \quad \text{for each } j = 1, \dots, \ell \\ \pi_{\mathbf{V}}(\mathbf{S}_1, \dots, \mathbf{S}_\ell) &= 0 \quad \text{where } \mathbf{S}_j := \mathbf{R}_{j,0} + \mathbf{R}_{j,j} + g_j^{(1)} \mathbf{I}_s \\ \mathbf{R}_{j,k} + \mathbf{R}_{k,j} + g_{j,k}^{(2)} \mathbf{I}_s &= 0 \quad \text{for } 1 \leq j \lesssim k \leq \ell, \end{aligned} \quad (19)$$

where \mathbf{I}_s is the s -dimensional identity matrix.

Proof. We follow a similar approach to the discussion presented in Eq. (15) and (16). Let us write $\mathbf{A}_{x_j, j} = \mathbf{A}_{0,j} + x_j \boldsymbol{\Delta}$, for each j , and substitute them into the expression of ρ in Lemma 1. As before, since we are working with the case where the variable x_j takes the value in \mathbb{F}_2 , we impose the condition that $x_j^2 = x_j$. Expanding the expression of ρ , we have the following:

$$\rho = \mathbf{R}_1 \mathbf{A}_{0,1} + \cdots + \mathbf{R}_\ell \mathbf{A}_{0,\ell} + \rho_\Delta \boldsymbol{\Delta},$$

where

$$\rho_\Delta = \sum_{j=1}^{\ell} (\mathbf{R}_{j,0} + \mathbf{R}_{j,j})x_j + \sum_{1 \leq j \neq k \leq \ell} (\mathbf{R}_{j,k} + \mathbf{R}_{k,j})x_jx_k + g(x_1 + \alpha_1, \dots, x_\ell + \alpha_\ell).$$

Since ρ should belong to $\text{span}(\mathbf{V})$ for any $\mathbf{A}_{0,j}$ and Δ , it should satisfy that $\mathbf{R}_j \mathbf{A}_{0,j} \in \text{span}(\mathbf{V})$, for each j , and $\rho_\Delta \Delta \in \text{span}(\mathbf{V})$.

The condition that $\mathbf{R}_j \mathbf{A}_{0,j} \in \text{span}(\mathbf{V})$ is equivalent to that

$$\pi_{\mathbf{V}}(\mathbf{R}_{j,1}, \dots, \mathbf{R}_{j,\ell}) = 0.$$

We notice that any elements in $\text{span}(\mathbf{V})$ contain only linear terms since we assumed that \mathbf{H} contains only linear queries. Thus, any terms of degree ≥ 2 in ρ_Δ should vanish. This condition is equivalent to that the polynomial g is of at most degree 2 and the quadratic terms of ρ_Δ are zeros, i.e.

$$\mathbf{R}_{j,k} + \mathbf{R}_{k,j} + g_{j,k}^{(2)} \mathbf{I}_s = 0 \quad \text{for } 1 \leq j \neq k \leq \ell.$$

Regarding the linear terms in ρ_Δ , it should satisfy the following:

$$\pi_{\mathbf{V}}(\mathbf{S}_1, \dots, \mathbf{S}_\ell) = 0,$$

where each $\mathbf{S}_j = \mathbf{R}_{j,0} + \mathbf{R}_{j,j} + g_j^{(1)} \mathbf{I}_s$ is the coefficient of x_j in ρ_Δ . Therefore, we have proved our claims. \square

The following theorem is a straightforward consequence of the preceding lemmata. It provides guidance on selecting the control matrices to ensure the correctness of a garbling scheme.

Theorem 2. *Assume that the vector \mathbf{H} consists of only linear queries and the matrix \mathbf{M} is of full rank in the garbling equation \mathcal{G}_g . Then the garbling scheme Π associated with the garbling equation \mathcal{G}_g is correct, if and only if the followings hold:*

1. *The degree of the target gate g is less than or equal to 2;*
2. *The control matrix \mathbf{R}_j satisfies Eq. (19) for each j .*

Remark 2. From Theorem 2, we observe that if solely linear queries are permitted to the random oracle, then it becomes unfeasible to garble a gate of degree greater than or equal to 3 using the described method. Interestingly, this finding is equivalent to Corollary 4 in [1], which demonstrates the impossibility of garbling a higher fan-in gate of degree ≥ 3 .

Indeed, although Theorem 2 implies that garbling high-degree gates using solely linear queries is infeasible, the prospect of constructing a garbling scheme for such gates through *non-linear queries* remains open. Exploring this avenue could yield fascinating results and warrants further investigation. \square

As demonstrated in the example of the construction by Ashur et al. [1] in Sect. 4.2, choosing the control matrices in a manner that guarantees the correctness of a garbling scheme does not invariably imply that the scheme is also *private*. In other words, despite ensuring the correctness, the corresponding control matrices may inadvertently reveal some information about the permute bits. Consequently, we will explore the conditions under which information regarding the permute bits is compromised.

Precisely, we provide the following lemma and theorem:

Lemma 3. *For each j , suppose that there exists a non-zero vector \mathbf{v}_j such that*

$$\mathbf{v}_j^\top \mathbf{P}_j \neq 0 \text{ and } \mathbf{v}_j^\top \mathbf{P}_i = 0 \text{ for all } i \neq j.$$

Then, given $(\mathbf{R}_1(i_1, \dots, i_\ell), \dots, \mathbf{R}_\ell(i_1, \dots, i_\ell))$ for some $i_1, \dots, i_\ell \in \mathbb{F}_2$, one can compute the value $\tilde{g}_j(\alpha_1, \dots, \alpha_\ell)$ for each j , where \tilde{g}_j is a function that is defined by the following:

$$\tilde{g}_j := g_j^{(1)} + \sum_{k \lesssim j} i_k g_{k,j}^{(2)} + \sum_{j \lesssim k} i_k g_{j,k}^{(2)}.$$

Proof. First, we consider the case of $j = 1$. From the relations in Eq. (14), we have the following:

$$\begin{aligned} & \pi(\mathbf{S}_1, \dots, \mathbf{S}_\ell) + (i_1 + 1)\pi(\mathbf{R}_{1,1}, \dots, \mathbf{R}_{i,\ell}) + i_2\pi(\mathbf{R}_{2,1}, \dots, \mathbf{R}_{2,\ell}) + \dots + i_\ell\pi(\mathbf{R}_{\ell,1}, \dots, \mathbf{R}_{\ell,\ell}) \\ &= \pi(\mathbf{R}_{1,0} + i_1\mathbf{R}_{2,1} + \dots + i_\ell\mathbf{R}_{\ell,1} + g_1^{(1)}\mathbf{I}_s, * , \dots, *) \\ &= \pi(\tilde{\mathbf{R}}_1 + \tilde{g}_1\mathbf{I}_s, * , \dots, *) \\ &= 0, \end{aligned}$$

where $\tilde{\mathbf{R}}_k := \mathbf{R}_k(i_1, \dots, i_\ell)$. In the second equality, we used the relation that $\mathbf{R}_{j,k} = \mathbf{R}_{k,j} + g_{j,k}^{(2)}\mathbf{I}_s$ from Eq. (14).

Multiplying the vector \mathbf{v}_1^\top to the both sides yields

$$\mathbf{v}_1^\top \mathbf{P}_1 \mathbf{R}_1(i_1, \dots, i_\ell) = \tilde{g}_1 \mathbf{v}_1^\top \mathbf{P}_1.$$

Thus, calculating the left-hand side, one can compute the value of \tilde{g}_1 . Similar arguments hold for the other cases. □

The following theorem is an immediate consequence of the preceding lemma. It specifies the conditions under which the garbling scheme leaks information about the permute bits.

Theorem 3. *For each j , suppose that there exists a non-zero vector \mathbf{v}_j such that*

$$\mathbf{v}_j^\top \mathbf{P}_j \neq 0 \text{ and } \mathbf{v}_j^\top \mathbf{P}_i = 0 \text{ for all } i \neq j.$$

Then the garbling scheme with the garbling equation \mathcal{G}_g cannot be privately garbleable.

Proof. Recall that the evaluator is given $(\mathbf{R}_1(i_1, \dots, i_\ell), \dots, \mathbf{R}_\ell(i_1, \dots, i_\ell))$ for some $i_1, \dots, i_\ell \in \mathbb{F}_2$ depending on her choices of the active input labels. By Lemma 3, she can compute the value of $\tilde{g}_j(\alpha_1, \dots, \alpha_\ell)$ for each $j = 1, \dots, \ell$. Thus, the garbling scheme leaks some information on the permute bits $\alpha_1, \dots, \alpha_\ell$. It violates the privacy property. \square

5.3 Discussions

In this section, we discuss on several interesting implication of our results.

RR21 Construction. We observe that our proposed attack does not apply to the construction by Rosulek and Roy [15], implying that it remains secure against our presented attack. Recall that in their construction the matrix \mathbf{W} is of the form $\mathbf{W} = \mathbf{W}_1x + \mathbf{W}_2y$, where:

$$\mathbf{W}_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{W}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Hence, the cokernel of \mathbf{W} is $\text{coker}(\mathbf{W}) = \mathbf{P} = [\mathbf{P}_1 \mid \mathbf{P}_2] = [0 \ 1 \mid 1 \ 0]$. Since the matrix \mathbf{P}_k consists of only one row, its left kernel is trivial. In other words, there exists no non-zero vector \mathbf{v} such that $\mathbf{v}^\top \mathbf{P}_k = 0$, which precludes our proposed attack.

Indeed, as observed in the explicit formula presented in Sect. 3, the values $\mathbf{R}_A(i, j)$ and $\mathbf{R}_B(i, j)$ do not reveal any information on the permute bits α and β . Due to the enough degrees of freedom available in the choice of the control matrices, the permute bits are effectively masked by these random values, hindering the evaluator from deducing the permute bits.

When Our Attack Works. Recalling that the matrix \mathbf{P}_i depends directly on the matrix $\mathbf{V} = [\mathbf{I} \mid \mathbf{W}]$, we will now present a simple criterion for determining whether our attack can be applied simply by observing the matrix \mathbf{W} . Recall that $\mathbf{W} = \mathbf{W}_1x_1 + \dots + \mathbf{W}_\ell x_\ell$, where each \mathbf{W}_j is a $(s \times r)$ -binary matrix. We assume that $s \leq r$; we will address later why this assumption seems to hold whenever the matrix \mathbf{V} includes the identity matrix \mathbf{I} .

In what follows, we argue that if the \mathbf{W}_j is not of a full rank for some j , then our attack is applicable. Without loss of generality, assume that the rank of \mathbf{W}_1 is less than s . By this assumption, the rows of \mathbf{W}_1 are linearly dependent. Therefore, there exists a non-zero vector \mathbf{p}_1 such that $\mathbf{p}_1^\top \mathbf{W}_1 = 0$. Since $[\mathbf{p}_1^\top \mid 0 \mid \dots \mid 0]$ is an element of the cokernel of $[\mathbf{W}_1^\top \mid \dots \mid \mathbf{W}_\ell^\top]^\top$ and the matrix \mathbf{P} is a basis matrix of the cokernel, there exists a non-zero vector \mathbf{v}_1 such that $\mathbf{v}_1 \mathbf{P} = [\mathbf{v}_1 \mathbf{P}_1 \mid \dots \mid \mathbf{v}_1 \mathbf{P}_\ell] = [\mathbf{p}_1^\top \mid 0 \mid \dots \mid 0]$. Thus, the vector \mathbf{v}_1 satisfies the condition in Theorem 3.

For instance, as discussed in Sect. 4, in the construction by Ashur et al. [1], the (3×4) -matrix \mathbf{W}_j has rank 2. Hence, their construction is vulnerable to our attack.

Necessary Conditions to Succeed Our Attack. Unfortunately, the condition in Theorem 3 is not a necessary condition for our attack to succeed. We will now examine the following intriguing case study. Let us consider a three-sliced garbling scheme, as described in the construction presented by Ashur et al., i.e. we set $s = 3$. We maintain the same notation used in Sect. 4.2. In our example, let us take into account the following linear combinations of linear queries:

$$\begin{aligned} D^1 + g_{tri}(u, v, w)\Delta^1 &= H(B_y + C_z) + H(A_x + B_y + C_z) + \dots \\ D^2 + g_{tri}(u, v, w)\Delta^2 &= H(A_x + C_z) + H(A_x + B_y + C_z) + \dots \\ D^3 + g_{tri}(u, v, w)\Delta^3 &= H(A_x + B_y) + H(A_x + B_y + C_z) + \dots \end{aligned}$$

In this case, we have the matrix $\mathbf{V} = [\mathbf{I} \mid \mathbf{W}]$ where

$$\mathbf{W} = \begin{bmatrix} y + z & 0 & 0 & x + y + z \\ 0 & x + z & 0 & x + y + z \\ 0 & 0 & x + y & x + y + z \end{bmatrix} = \mathbf{W}_1x + \mathbf{W}_2y + \mathbf{W}_3z.$$

One might check that each matrix \mathbf{W}_j has full rank. Moreover, we could not find a vector \mathbf{v}_j satisfying the condition in Theorem 3. However, if we find the control matrices fulfilling Eq. (19), we observe that the corresponding garbling scheme still leaks some information on the permute bits. Precisely, when the evaluator is given the control matrices at $(x, y, z) = (0, 0, 0)$, i.e. the active input labels are A_0, B_0, C_0 , the evaluator can deduce the value of $\beta + \gamma$ by comparing the values $\mathbf{R}_{1,0} = \mathbf{R}_A(0, 0, 0)$ and $\mathbf{R}_{2,0} = \mathbf{R}_B(0, 0, 0)$.

Consequently, it has been observed that the rank condition on \mathbf{W}_j is insufficient to construct a secure garbling scheme. It remains as an open problem to explore further when a secure garbling scheme can be constructed.

A How to Randomize the Control Bits

In this section, we explain how the dicing technique works from the algebraic perspective. For the sake of readability, we mainly describe the technique with the example of RR21’s construction.

At the beginning of the dicing technique, the garbler chooses $(\mathbf{R}_A, \mathbf{R}_B)$ at random among 2^{14} possible choices. Assume that the choice is

$$\mathbf{R} = [\mathbf{R}_A \mid \mathbf{R}_B] = \left[\begin{array}{cc|cc} 0 & 0 & x + \alpha & y + \beta + 1 \\ 0 & 0 & y & x \end{array} \right].$$

It is chosen by setting all the free variables zero except $e_3 = 1$.

To send the information on \mathbf{R} , the garbler encrypts it column by column. More precisely, say $\mathbf{R} = [\vec{r}_1, \dots, \vec{r}_4]$, where \vec{r}_k is the k -th column of \mathbf{R} . The garbler makes random oracle queries and define

$$\vec{S}^{con} := (H^c(A^0), H^c(A^1), H^c(B^0), H^c(B^1), H^c(A^0 + B^0), H^c(A^0 + B^1))^T,$$

where H^c is a random oracle that returns an 1-bit string (it is usually chosen as the least significant bit of outputs by the random oracle).

Given the column \vec{r}_k for each k , choose $\vec{z}_k := (z_{k1}, \dots, z_{k5})^\top$ such that

$$\mathbf{V}\vec{z}_k = \mathbf{M}\vec{S}^{con} + \vec{r}_k. \tag{20}$$

Then it returns the vector \vec{z}_k which comprises the ciphertexts encrypting \vec{r}_k .

For instance, let us take an example of $\vec{r}_3 = (x + \alpha, y)^\top$. By comparing both sides, we have

$$\begin{aligned} z_{31} &= H^c(A^0) + H^c(A^0 + B^0) + \alpha \\ z_{32} &= H^c(B^0) + H^c(A^0 + B^0) \\ z_{33} &= H^c(A^0) + H^c(A^1) + 1 \\ z_{34} &= H^c(B^0) + H^c(B^1) + 1 \\ z_{35} &= H^c(A^0 + B^0) + H^c(A^0 + B^1). \end{aligned}$$

Let \mathbf{V}_{ij} be the value of \mathbf{V} evaluated at $(x, y) = (i, j)$. Upon receiving \vec{z}_k , on input A^i and B^j , the evaluator computes

$$\vec{r}_k = \mathbf{V}_{ij}\vec{z}_k + \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} H(A^i) \\ H(B^j) \\ H(A^i + B^j) \end{bmatrix}.$$

It is easily verified that \vec{r}_k is the value of \vec{r}_k evaluated at $(x, y) = (i, j)$.

We observe that the above argument works in general not only for RR21's construction. Actually, the control bit randomization is carried out by encrypting each columns of \mathbf{R} , the randomly chosen control bits. Moreover, it is encrypted via *the same garbling equation* as that used for the original garbling construction. In other words, the matrices \mathbf{M} and \mathbf{V} in Eq. (20) are the same as the original garbling equation. The only condition for the control bits encryption to work, it suffices to see whether \vec{r}_k belongs to the same space spanned by the columns of \mathbf{M} or \mathbf{V} . And it turns out to be equivalent that \vec{r}_k satisfies the relation $\pi_{\mathbf{V}}$ in Sect. 5. Recall that \vec{r}_k is the column of \mathbf{R} . We observe that \mathbf{R} , thus each of its columns, satisfies the relation $\pi_{\mathbf{V}}$ which is the desired result. Henceforth, we argue that the control bit randomization is always possible with its original garbling equation.

To help readers' understanding, let us call back the previous example of the RR21 construction. In this case, the relation π is equivalent to say that the y -coefficient on the top is the same as the x -coefficient of the bottom. We see that, for each \vec{r}_k , it satisfies the condition.

Let us consider why this technique does not reveal the information on α and β . We see that the entire value of \mathbf{R} will definitely disclose the permute bits. Observe that \vec{z}_k 's are encrypting the coefficients of the polynomials in \mathbf{R} using \vec{S}^{con} . And the decryption only reveals the value of the polynomials in \mathbf{R} evaluated at $(x, y) = (i, j)$. Without knowing the wire labels other than A_i and B_j , the evaluator cannot evaluate the polynomials outside of (i, j) . Thus, it does not disclose the entire information on \mathbf{R} .

One might observe that the number of additional ciphertexts required to encrypt the control matrices can be further reduced in RR21’s construction. Let us consider the control matrices given by

$$[\mathbf{R}_A \mid \mathbf{R}_B] = \left[\begin{array}{cc|cc} r_1 & r_2 & r_2 + x & r_1 + r_2 + y \\ r_2 & r_1 + r_2 & r_1 + r_2 & r_1 + x \end{array} \right]$$

where $r_1(x, y) := \alpha x + (\beta + 1)y + c$ and $r_2(x, y) := (\beta + 1)x + (\alpha + \beta + 1)y + e$ are the polynomials in $\mathbb{F}_2[x, y]$, and the bits c and e are randomly chosen. It can be readily verified that the above control matrices yield a correct garbling scheme for RR21’s construction. Thus, it is enough to send only the encryption of $(r_1, r_2)^\top$, instead of sending entire encryptions of all columns. Therefore, it reduces the number of ciphertexts garbling the control bits.

B How to Choose Control Matrices

Given the matrix \mathbf{V} as defined in Eq. (13), for any vector $\boldsymbol{\nu} = \boldsymbol{\nu}_0 + \boldsymbol{\nu}_1x + \boldsymbol{\nu}_2y + \boldsymbol{\nu}_3z \in \text{span}(\mathbf{V})$, we obtain $\pi_{\mathbf{V}}(\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \boldsymbol{\nu}_3) = \mathbf{P}_1\boldsymbol{\nu}_1 + \mathbf{P}_2\boldsymbol{\nu}_2 + \mathbf{P}_3\boldsymbol{\nu}_3 = 0$, where

$$\mathbf{P} = [\mathbf{P}_1 \mid \mathbf{P}_2 \mid \mathbf{P}_3] = \left[\begin{array}{ccc|ccc|ccc} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{array} \right]. \tag{21}$$

By computing the control matrices satisfying Eq. (15) and (16), we can provide their explicit formula as follows.

B.1 Formulas for the Control Matrices

We provide explicit formulas for the control matrices.

$$\begin{aligned} \mathbf{R}_A &= \begin{bmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \\ a_0 + \beta + \gamma & b_0 & c_0 + \beta + \gamma \end{bmatrix} + \begin{bmatrix} a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \\ a_3 & b_3 & c_3 \end{bmatrix} x \\ &+ \begin{bmatrix} a_4 + 1 & b_4 & c_4 + 1 \\ a_4 + 1 & b_4 & c_4 + 1 \\ a_4 & b_4 & c_4 \end{bmatrix} y + \begin{bmatrix} a_4 & b_4 & c_4 \\ a_4 + 1 & b_4 & c_4 + 1 \\ a_4 + 1 & b_4 & c_4 + 1 \end{bmatrix} z \\ \mathbf{R}_B &= \begin{bmatrix} d_0 & e_0 & f_0 \\ d_0 + \alpha & e_0 + \alpha & f_0 \\ a_1 + 1 & b_1 + \beta + \gamma + 1 & c_1 + \alpha + 1 \end{bmatrix} + \begin{bmatrix} a_4 & b_4 & c_4 + 1 \\ a_4 + 1 & b_4 + 1 & c_4 + 1 \\ a_4 & b_4 & c_4 + 1 \end{bmatrix} x \\ &+ \begin{bmatrix} d_5 & e_5 & f_5 \\ d_5 & e_5 & f_5 \\ a_4 + 1 & b_4 + 1 & c_4 + 1 \end{bmatrix} y + \begin{bmatrix} a_4 + 1 & b_4 + 1 & c_4 + 1 \\ a_4 + 1 & b_4 + 1 & c_4 + 1 \\ a_4 + 1 & b_4 + 1 & c_4 + 1 \end{bmatrix} z \end{aligned}$$

$$\begin{aligned}
\mathbf{R}_C = & \begin{bmatrix} a_1 + \alpha + 1 & b_1 + \beta + \gamma + 1 & c_1 + 1 \\ g_1 & h_1 & i_1 \\ g_1 & h_1 + \alpha & i_1 + \alpha \end{bmatrix} + \begin{bmatrix} a_4 + 1 & b_4 & c_4 \\ a_4 + 1 & b_4 + 1 & c_4 + 1 \\ a_4 + 1 & b_4 & c_4 \end{bmatrix} x \\
& + \begin{bmatrix} a_4 + 1 & b_4 + 1 & c_4 + 1 \\ a_4 + 1 & b_4 + 1 & c_4 + 1 \\ a_4 + 1 & b_4 + 1 & c_4 + 1 \end{bmatrix} y + \begin{bmatrix} a_4 + 1 & b_4 + 1 & c_4 + 1 \\ g_6 & h_6 & i_6 \\ g_6 & h_6 & i_6 \end{bmatrix} z, \quad (22)
\end{aligned}$$

where all of the entries are binary elements. We observe that the set of the pairs of $(\mathbf{R}_A, \mathbf{R}_B, \mathbf{R}_C)$ is isomorphic to 24-dimensional subspace.

References

1. T. Ashur, C. Hazay, and R. Satish. On the feasibility of sliced garbling. Cryptology ePrint Archive, Paper 2024/389, 2024. <https://eprint.iacr.org/2024/389>.
2. C. Baek and T. Kim. Can we beat three halves lower bound?: (im)possibility of reducing communication cost for garbled circuits. Cryptology ePrint Archive, Paper 2024/803, 2024. <https://eprint.iacr.org/2024/803>.
3. M. Ball, H. Li, H. Lin, and T. Liu. New ways to garble arithmetic circuits. In C. Hazay and M. 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*, volume 14005 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2023.
4. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In H. Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM, 1990.
5. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In T. Yu, G. Danezis, and V. D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796. ACM, 2012.
6. L. Fan, Z. Lu, and H. Zhou. Column-wise garbling, and how to go beyond the linear model. *IACR Cryptol. ePrint Arch.*, page 415, 2024.
7. S. Gueron, Y. Lindell, A. Nof, and B. Pinkas. Fast garbling of circuits under standard assumptions. In I. Ray, N. Li, and C. Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 567–578. ACM, 2015.
8. D. Heath and V. Kolesnikov. One hot garbling. In Y. Kim, J. Kim, G. Vigna, and E. Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 574–593. ACM, 2021.
9. D. Heath, V. Kolesnikov, and L. K. L. Ng. Garbled circuit lookup tables with logarithmic number of ciphertexts. In M. Joye and G. Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 185–215. Springer, 2024.

10. C. Kempka, R. Kikuchi, and K. Suzuki. How to circumvent the two-ciphertext lower bound for linear garbling schemes. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 967–997, 2016.
11. V. Kolesnikov, P. Mohassel, and M. Rosulek. Flexor: Flexible garbling for XOR gates that beats free-xor. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457. Springer, 2014.
12. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfssdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.
13. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In S. I. Feldman and M. P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.
14. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.
15. M. Rosulek and L. Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In T. Malkin and C. Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 94–124. Springer, 2021.
16. A. C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.
17. S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2015.



Revisiting OKVS-Based OPRF and PSI: Cryptanalysis and Better Construction

Kyoohyung Han¹, Seongkwang Kim^{1(✉)}, Byeonghak Lee¹, and Yongha Son^{2(✉)}

¹ Samsung SDS, Seoul, Korea

{kh89.han, sk39.kim, byghak.lee}@samsung.com

² Sungshin Women's University, Seoul, Korea

yongha.son@sungshin.ac.kr

Abstract. Oblivious pseudorandom function (OPRF) is a two-party cryptographic protocol that allows the receiver to input x and learn $F(x)$ for some PRF F , only known to the sender. For private set intersection (PSI) applications, OPRF protocols have evolved to enhance efficiency, primarily using symmetric key cryptography. Current state-of-the-art protocols, such as those by Rindal and Schoppmann (Eurocrypt '21), leverage vector oblivious linear evaluation (VOLE) and oblivious key-value store (OKVS) constructions.

In this work, we identify a flaw in an existing security proof, and present practical attacks in the malicious model, which results in additional PRF evaluations than the previous works' claim. In particular, the attack for malicious model is related to the concept of OKVS overfitting, whose hardness is conjectured in previous works. Our attack is the first one to discuss the concrete hardness of OKVS overfitting problem.

As another flavour of contribution, we generalize OKVS-based OPRF constructions, suggesting new instantiations using a VOLE protocol with only Minicrypt assumptions. Our generalized construction shows improved performance in high-speed network environments, narrowing the efficiency gap between the OPRF constructions over Cryptomania and Minicrypt.

Keywords: oblivious pseudorandom function · oblivious key-value store · private set intersection

1 Introduction

Oblivious pseudo random function (OPRF) is a two-party cryptographic protocol that allows the receiver to input x and learn $F(x)$ for some PRF F , only known to the sender. OPRF in general can be thought of as a two-party protocol where the sender inputs a secret key and the receiver inputs some items to be evaluated and obtains a PRF value for each input. In contrast, OPRF for PSI application – which will be called *batch OPRF* – have been advanced in an independent

Y. Son—This work was done while Y. Son was at Samsung SDS.

© International Association for Cryptologic Research 2025

K.-M. Chung and Y. Sasaki (Eds.): ASIACRYPT 2024, LNCS 15491, pp. 266–296, 2025.

https://doi.org/10.1007/978-981-96-0944-4_9

direction. As two-party PSI does not require the sender to choose the secret key nor to evaluate each item separately, batch OPRF usually outputs a random secret key to the sender and batched PRF evaluations to the receiver. Any item which is not in the batched inputs at the moment of invoking the OPRF protocol cannot be evaluated.

At the cost of such demerits, batch OPRFs are concretely efficient since they heavily use symmetric key cryptography rather than public key cryptography such as Diffie-Hellman computation. Pinkas et al. [9] proposed a PSI protocol based on a special sort of data structure called oblivious key-value store (OKVS) (called PaXoS at that time of writing). Although this paper does not include any explicit OPRF construction, the PSI protocol can be naturally extend to a PSI protocol, which we call PRTY construction hereafter. After then, Rindal and Schoppmann proposed an improved construction while replacing a subroutine of the PRTY construction by another functionality called vector-oblivious linear evaluation (VOLE), which forms the state-of-the-art protocols of (batch) OPRF and PSI protocols [3, 10], and we call this by RS construction.

Currently, the RS construction that utilizes VOLE over large field such as $\text{GF}(2^{128})$ has much better performance than the PRTY construction, in both computation and communication view. It depends on so-called VOLE protocols based on pseudorandom correlation generator (PCG) [4, 5, 11], which assumes some (some variants of) Learning Parity with Noise (LPN). Meanwhile, the PRTY construction used the OOS functionality [8] (or $\text{GF}(2)$ -VOLE), which can be realized with only Minicrypt assumptions. However, as the performance gap between two constructions is fairly large currently, the advantage of PRTY construction in robustness may seem less attractive.

In batch OPRF protocols based on OKVS, the receiver encodes all its input items into an OKVS to obtain the PRF values of them. Because of the linearity of existing OKVS, the OPRF protocols based on OKVS by nature allows more evaluations than it should. Pinkas et al. upper bound the allowed number of evaluations information-theoretically, and Garimella et al. formalize the problem to overpack in an OKVS as *OKVS overfitting problem*.

1.1 Our Contribution

We revisit the security of batch OPRFs, with respect to the number of evaluation. Although OPRF protocols should prevent the receiver to arbitrarily evaluate the PRF value, we found that batch OPRF protocols in [3, 10, 13] based on OKVS allow more evaluations than the authors claimed. We point out the flaw in the security proof, and present practical attacks on them.

In the malicious model, we propose an overfitting algorithm to solve the OKVS overfitting problem, whose main idea is to reduce the overfitting problem of OKVS to either a k -XOR problem or a multicollision finding problem. To the best of our knowledge, the second attack is the first constructive (not information-theoretical) solving algorithm for the OKVS overfitting problem [7]. To prevent these attacks, [13] in the malicious setting incurs 1371% communication overhead, where the number of items is set to 2^{20} .

In PSI applications, our attacks usually incur a situation that a corrupt receiver knows “the sender does not have some specific items”. In the literature, the security of PSI only refers to the opposite side (“the sender have some specific items”). However, we clarify that such situation may leak some membership information by computing statistical distance. Depends on the application setting, the distance may exceed $2^{-\lambda}$ where λ is the statistical security parameter.

In a construction aspect, we suggest some possible mitigations to prevent our proposed attacks, and also provide revised security proof along with the mitigations in the malicious model. The mitigations shows more efficiency compared to solely following the PRTY information-theoretic bound.

We also investigate more general instantiations of OKVS-based OPRF construction, and suggest new instantiations that generalize the PRTY construction [9] using a recently proposed VOLE protocol over Minicrypt assumptions [14]. In the fast network environments, our generalization even outperforms the RS construction (with quasi-cyclic code). In slower network environment, as our generalization requires more communication than the RS constructions, the RS constructions remains the best one. Although, our generalization is still meaningful in a view that it narrows the communication gap between the RS construction (Cryptomania) and the PRTY construction (Minicrypt) from $4.2\times$ to $1.3\times$.

2 Preliminaries

2.1 Notation

For a matrix A , each i -th row vector is denoted by \vec{A}_i and j -th column vector is denoted by \vec{A}^j . For a vector $\vec{\Delta} = (\Delta_1, \dots, \Delta_n)$ and a matrix U , we denote $\vec{\Delta} \odot U := (\Delta_1 \cdot \vec{U}^1, \dots, \Delta_{n_c} \cdot \vec{U}^{n_c})$. Unless otherwise stated, every field \mathbb{F} in our paper is assumed to be of characteristic 2, or we explicitly write the finite field (or Galois field) of size q by $\text{GF}(q)$. For a field \mathbb{F} , we write a linear code C in a k_c dimensional subspace in \mathbb{F}^{n_c} with minimum distance d_c by $[n_c, k_c, d_c]_{\mathbb{F}}$, with an exception of $[n_c, k_c, d_c]_2$ for $\mathbb{F} = \text{GF}(2)$ case. We often consider a $[n_c, k_c, d_c]_{\mathbb{F}}$ linear code as a map $C : \mathbb{F}^{k_c} \rightarrow \mathbb{F}^{n_c}$ to write the codeword on $\vec{x} \in \mathbb{B}^{k_c}$ by $C(\vec{x})$. We denote computational security parameter by κ , and statistical security parameter by λ . We denote $B(n, p)$ binomial distribution of probability p and n independent experiments.

2.2 Vector Oblivious Linear Evaluation

A (subfield) vector oblivious linear evaluation, precisely (\mathbb{F}, \mathbb{B}) -VOLE outputs two parties a secret shares of scalar-vector multiplication $\Delta \cdot \vec{U}$ for randomly chosen $\Delta \in \mathbb{F}$ and $\vec{U} \in \mathbb{B}^m$ to each party. When the subfield \mathbb{B} equals to \mathbb{F} , we call it simply by \mathbb{F} -VOLE. The corresponding ideal functionality $\mathcal{F}_{\text{vole}}$ is defined as Fig. 1.

During the past few years, the performance of VOLE for large fields such as $\text{GF}(2^{128})$ has been rapidly improved, thanks to the advances on pseudorandom

correlation generator (PCG) [5] based on learning with parity (LPN) problem. However, such rapid performance is enabled by assuming the hardness of LPN over somewhat non-standard codes [4, 6, 11]. Indeed, there has been proposed an attack [11] on the silver code utilized in [6]. Meanwhile, Roy [14] proposed a novel VOLE protocol that only requires Minicrypt assumptions, unlike the LPN-based protocols. It is practically efficient for small field such as $\text{GF}(2^f)$ with $f \leq 8$.

Parameters: Two parties (a sender and a receiver). A field \mathbb{F} with a subfield \mathbb{B} , and an integer m representing the length of output vector.

Functionality:

- If the receiver is malicious, wait for them to send $\vec{V} \in \mathbb{F}^m$ and $\vec{U} \in \mathbb{B}^m$. Then sample $\Delta \leftarrow \mathbb{F}$ and let $\vec{W} := \vec{V} + \Delta \cdot \vec{U} \in \mathbb{F}^m$.
- If the sender is malicious, wait for them to send $\vec{W} \in \mathbb{F}^m$ and $\Delta \in \mathbb{F}$. Then sample $\vec{U} \leftarrow \mathbb{B}^m$ and let $\vec{V} := \vec{W} - \Delta \cdot \vec{U} \in \mathbb{F}^m$.
- Otherwise, i.e., adversarial party is semi-honest, sample $\vec{V} \leftarrow \mathbb{F}^m, \vec{U} \leftarrow \mathbb{B}^m$ and $\Delta \leftarrow \mathbb{F}$, and let $\vec{W} = \vec{V} + \Delta \cdot \vec{U} \in \mathbb{F}^m$

Output $\vec{W} \in \mathbb{F}^m$ and $\Delta \in \mathbb{F}$ to the sender, and $\vec{V} \in \mathbb{F}^m$ and $\vec{U} \in \mathbb{B}^m$ to the receiver.

Fig. 1. An ideal functionality of $\mathcal{F}_{\text{vole}}(\mathbb{F}, \mathbb{B})$ for (\mathbb{F}, \mathbb{B}) -vector oblivious linear evaluation

2.3 Oblivious Key-Value Store

Informally, an oblivious key-value store (OKVS) is a data structure that efficiently encodes n pairs of keys and values, which satisfies if a value are random, the corresponding key cannot be recovered from the encoding of the key-value pairs.

Definition 1 (Oblivious Key-Value Store). *An oblivious key-value store (OKVS) with key universe \mathcal{K} and value universe \mathcal{V} consists of two functions:*

- $\text{Ecd} : (\mathcal{K} \times \mathcal{V})^n \rightarrow \mathcal{V}^m \cup \{\perp\}$, a function that receives n distinct key-value pairs $(k_i, v_i)_{i \in [n]}$ then outputs encoding S or failure symbol \perp ;
- $\text{Dcd} : \mathcal{V}^m \times \mathcal{K} \rightarrow \mathcal{V}$, a function that receives encoding S and a key k then outputs the associated value v .

For correctness, for all $I \subset \mathcal{K} \times \mathcal{V}$ of n elements with distinct keys and an ordering $\vec{I} \in (\mathcal{K} \times \mathcal{V})^n$ of I , an OKVS should satisfies

$$(k, v) \in I \text{ and } \text{Ecd}(\vec{I}) = S \neq \perp \Rightarrow \text{Dcd}(S, k) = v.$$

For obliviousness, for any pair of lists of n distinct keys $(k_1, \dots, k_n) \in \mathcal{K}^n$ and $(k'_1, \dots, k'_n) \in \mathcal{K}^n$ and n random values $v_1, \dots, v_n \leftarrow_{\S} \mathcal{V}$, $\text{Ecd}((k_1, v_1), \dots, (k_n, v_n))$ and $\text{Ecd}((k'_1, v_1), \dots, (k'_n, v_n))$ should be computationally indistinguishable.

The storage efficiency of OKVS can be measure by the expansion ratio of the number of key-value pairs n and the length of the encoding vector m . Many known OKVS constructions achieves $m = (1 + \varepsilon_{\text{okvs}}) \cdot n$ for some small constant $\varepsilon_{\text{okvs}}$: PaXoS [9] achieves $\varepsilon \approx 1.4$, 3H-GCT [7] and RR22 [10] achieve $\varepsilon \approx 0.3$, and RB-OKVS [3] achieve ε down to 0.03.

Several OKVS applications, such as OPRF and PSI, expect OKVS to have linearity and support for some sort of homomorphic operations, and we call such OKVS by *linear OKVS*.

Definition 2 (Linear OKVS). *An OKVS is linear if there exists a function $\text{row} : \mathcal{K} \rightarrow \mathcal{V}^m$ such that $\text{Dcd}(S, k) = \langle \text{row}(k), S \rangle$ for all $k \in \mathcal{K}$ and $S \in \mathcal{V}^m$. If the range of such function row can be restricted to a set of binary vectors, i.e., $\text{row} : \mathcal{K} \rightarrow \{0, 1\}^m$, we call the OKVS as binary linear, or simply binary.*

3 Oblivious PRF and OKVS-Based Constructions

This section provides a formal definition of oblivious PRF (and private set intersection), and reviews the OKVS-based OPRF constructions [9, 13], which consists of the state-of-the-art protocols [3, 10].

Parameters: Two parties (a sender and a receiver). The receiver’s set size parameters n for the semi-honest model and n' for the malicious model. An OPRF output length ℓ_2 .

Functionality: Upon the receiver’s input set X , abort if $|X| > n'$ when the receiver is malicious. The functionality defines a random function $F : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_2}$, and output $F(X) = \{F(x) \mid x \in X\}$ to the receiver. After then, upon the sender’s query y , the functionality sends $F(y)$ to the Sender.

Fig. 2. Ideal functionality $\mathcal{F}_{\text{oprf}}$ of oblivious pseudorandom function.

3.1 Ideal Functionalities and Generic PSI Construction

The ideal functionality of an oblivious pseudorandom function (OPRF) is described in Fig. 2, and the (two-party) private set intersection (PSI) functionality is described in Fig. 3. Ideally, the receiver should not be able to obtain any OPRF evaluations other than for its input. However, the OPRF definition in Fig. 2 allows more PRF evaluations, denoted by $n' > n$ for a malicious receiver, and the PSI definition also allows at most n' items for the malicious receiver. This reflects the fact that known OPRF (or PSI) protocols enable an adversary to learn OPRF values $F(X)$ (or $X \cap Y$) for some $n' (> n)$ -sized set X , while pretending to run the protocol with a size n set. This type of definition has been widely adopted in the literature [9, 13].

Parameters: Two parties (a sender and a receiver). The sender’s set size parameter n_y , and the receiver’s set size parameters n_x for the semi-honest model and n' for the malicious model.

Functionality: Upon the receiver’s input set X and the Sender’s input set Y , abort if $|X| > n'$ when the receiver is malicious, and outputs $X \cap Y$ to the receiver.

Fig. 3. Ideal functionality \mathcal{F}_{psi} of (2-party) private set intersection.

PSI from OPRF. Given an OPRF functionality $\mathcal{F}_{\text{oprf}}$, it is straightforward to construct a protocol that realizes \mathcal{F}_{psi} , as shown in Fig. 4. This protocol requires one additional round of communication, with $\ell_2 \cdot n_y$ bits transferred from the sender to the receiver, on top of the communication cost for realizing $\mathcal{F}_{\text{oprf}}$. The concrete choice of ℓ_2 has been improved using better security proofs, and our paper adapts the state-of-the-art result from [13].

Parameters: Two parties (a sender and a receiver). An OPRF functionality $\mathcal{F}_{\text{oprf}}$ with set size parameters n_x or n' , and OPRF length ℓ_2 .

Protocol: Upon an input set X from the receiver, and an input set Y from the sender, the protocol runs as follows:

1. The sender and the receiver interact with $\mathcal{F}_{\text{oprf}}$ with the receiver’s input set X .
2. As $\mathcal{F}_{\text{oprf}}$ outputs, the receiver obtains $F(X)$. Then the sender obtains $F(Y) = \{F(y) \in \{0, 1\}^{\ell_2} : y \in Y\}$ by querying each $y \in Y$ to $\mathcal{F}_{\text{oprf}}$.
3. The sender sends $F(Y)$ to the receiver in a random order, who outputs $Z = \{x \in X : F(x) \in F(Y)\}$.

Fig. 4. Protocol Π_{psi} for a private set intersection using $\mathcal{F}_{\text{oprf}}$.

Theorem 1 (Adapted from [13]). *The protocol Π_{psi} realizes the \mathcal{F}_{psi} functionality in the semi-honest model with $\ell_2 = \lambda + \log(n_x n_y)$ and in the malicious model with $\ell_2 = \kappa$, in a $\mathcal{F}_{\text{oprf}}$ -hybrid model.*

Remark 1. In more detail, the simulated view against the malicious sender in [13] is distinguishable from the real protocol execution with $2^{\ell_2}/n_x$ random oracle queries. Hence, the simulation with $\ell_2 = \kappa$ can be distinguished by fewer than 2^κ queries. Rindal and Schoppmann was already aware of this fact while they stucked to use $\ell_2 = \kappa$ [13]. To be more rigorous, it is required to set $\ell_2 = \kappa + \log n_x$ in order to simulate against the malicious sender. But we follow the choice $\ell_2 = \kappa$ as the distinguishing advantage does not lead to any actual information leakage.

3.2 OKVS-Based OPRF Constructions

We present an overview of the OKVS-based OPRF construction in Fig. 5, which captures the RS construction [3, 10, 13] with $\mathbb{F} = \text{GF}(2^{128})$ and the identity linear code $[1, 1, 1]_{\text{GF}(2^{128})}$, as well as the natural OPRF extension from the PRTY PSI protocol [7, 9] with $\mathbb{F} = \text{GF}(2)$ and non-trivial binary linear codes. Detailed parameter selections and security arguments will be presented in later sections, as one of our main contributions is to identify the flaws in previous works.

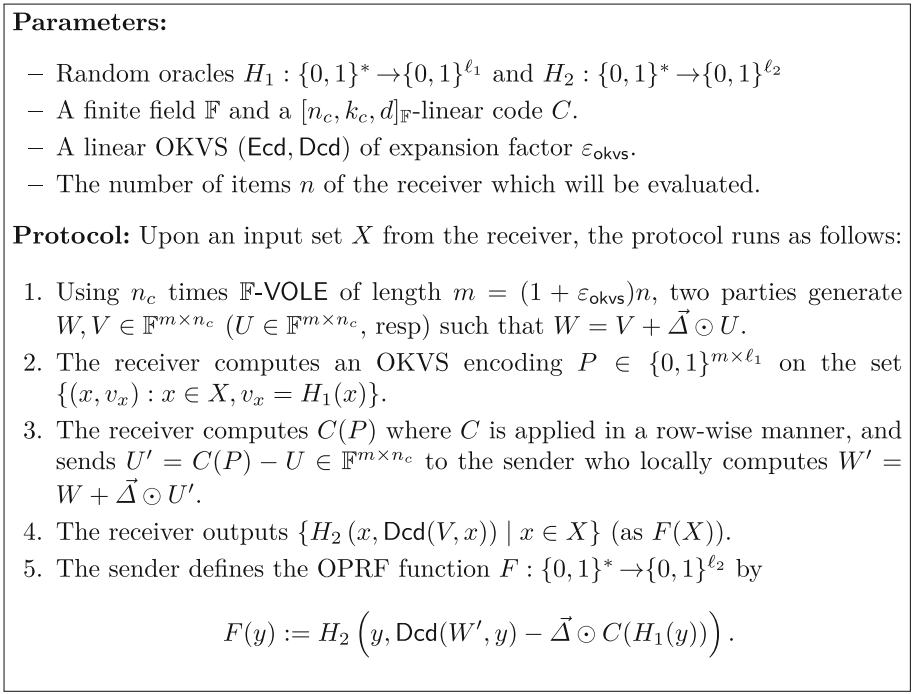


Fig. 5. Integrated overview of OKVS-based OPRF protocols.

To see the correctness, the set $\{H_2(x, \text{Dcd}(V, x)) \mid x \in X\}$ should equal to $F(X)$ with the sender's definition $F(y) = H_2(y, \text{Dcd}(W, y) - \vec{\Delta} \odot C(H_1(y)))$. From the linearity of Dcd and the linear code C , we have

$$\begin{aligned} F(y) &= H_2(y, \text{Dcd}(W', y) - \vec{\Delta} \odot C(H_1(y))) \\ &= H_2(y, \text{Dcd}(V, y) - \vec{\Delta} \odot C(\text{Dcd}(P, y) - H_1(y))) \end{aligned}$$

for any $y \in \{0, 1\}^*$. So, the OKVS correctness is ensured by the fact that $\text{Dcd}(P, x) = H_1(x)$ for every $x \in X$. Thus, we further have

$$F(x) = H_2(x, \text{Dcd}(V, y) - \vec{\Delta} \odot C(\text{Dcd}(P, y) - H_1(y))) = H_2(x, \text{Dcd}(V, x)). \quad (1)$$

Remark 2. The original description in [9] used OOS (correlated) OT extension [8] instead of VOLE over $\text{GF}(2)$, but two functionalities are exactly same.

Overfitted OKVS. The key to the correctness of OKVS-based OPRF in (1) is the equality $\text{Dcd}(P, x) = H(x)$ for every $x \in X$, ensured by the OKVS correctness. However, in OKVS-based OPRF protocols, a malicious receiver can arbitrarily generate the OKVS encoding P , allowing more than n items x such that $\text{Dcd}(P, x) = H_1(x)$. This enables the receiver to obtain the PRF values for these x . Garimella et al. [7] formalize this issue as the *OKVS overfitting game* as follows.

Definition 3 (*(n, n') -OKVS overfitting game, [7]*). *Let (Ecd, Dcd) be an OKVS with parameters chosen to support n items, and let $H_1 : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_1}$ be a random oracle. For any arbitrary PPT adversary \mathcal{A} that outputs $P \in \{0, 1\}^{\ell_1 \times m} \leftarrow \mathcal{A}^H(1^\kappa)$, define*

$$X' = \{x \mid \mathcal{A} \text{ queried } H_1 \text{ at } x \text{ and } \text{Dcd}(P, x) = H_1(x)\}.$$

If $|X'| > n'$, then the adversary wins the (n, n') -OKVS overfitting game.

We say the (n, n') -OKVS overfitting problem is hard for an OKVS construction if no PPT adversary wins this game except with negligible probability.

PRTY Bound. It can be easily observed that if the underlying OKVS is linear, a malicious receiver can obtain $m = (1 + \varepsilon_{\text{okvs}}) \cdot n$ PRF evaluations with almost no computational overhead. This fact leads OKVS-based OPRF protocols to focus on $n' = c \cdot m$ for some $c > 1$.

In [9], Pinkas et al. analyzed the choice of ℓ_1 (the output bit-length of H_1 in Fig. 5) that makes the OKVS-based OPRF construction information-theoretically secure against malicious adversaries. This analysis can be rephrased using OKVS terminology as follows.

Lemma 1 (PRTY bound, [9]). *Suppose an adversary makes q queries to random oracle H_1 with output length ℓ_1 , and then generates an OKVS P of size m . For a fixed integer n' , let \mathcal{E} denote the event that $\text{Dcd}(P, x) = H_1(x)$ for at least n' values x that were queried to H_1 . Then,*

$$\Pr[\mathcal{E}] \leq \frac{\binom{q}{n'}}{2^{(n'-m)\ell_1}}.$$

Suppose there is a linear system $Ax = b$ where $A \in (\mathbb{F}_{2^{\ell_1}})^{n' \times m}$ ($n' > m$) is invertible and $b \leftarrow_{\S} (\mathbb{F}_{2^{\ell_1}})^m$. The probability that b produces a solvable system is $2^{-(n'-m)\ell_1}$. Since the number of n' -tuples of queried items is $\binom{q}{n'}$, the bound in Lemma 1 is quite tight, with only a small gap possible due to a non-invertible OKVS system.

4 Security Flaws of OKVS-Based OPRFs

The OKVS-based OPRF naturally allows the receiver to locally compute $F(x)$ that satisfies $\text{Dcd}(P, x) = H_1(x) \in \{0, 1\}^{\ell_1}$. Thus, the length ℓ_1 should be set properly to prevent unwanted PRF evaluations. Indeed, previous OKVS-based OPRF protocols suggested some appropriate choice of ℓ_1 to bound the number of PRF evaluations by n (semi-honest) or n' (malicious), along with corresponding security arguments. However, in this section, we present some (possible) vulnerabilities in the previous works setting on n, n' and ℓ_1 , and show that it indeed implies more number of PRF evaluations than the works claimed.

4.1 Caution for Possible Semi-honest OPRF

Although previous works [9, 10, 13] claimed the malicious security of the OPRF protocol or the semi-honest security of the PSI protocol rather than the semi-honest security of OPRF protocol, one can naturally derive a semi-honest version of OPRF protocol and try to use it. In this section, we briefly point out which security issue can be popped up.

All semi-honest OKVS-based PSI protocols set $\ell_1 = \lambda + 2 \log n$ where n is the number of items in both parties, whose rationale is to prevent unwanted collision of hashed values. However, as random oracle is an idealization of cryptographic hash function, it should be assumed that the receiver is free to query to the random oracle before or after the protocol. It implies that the probability of the equality $\text{Dcd}(P, x) = H_1(x)$ for a fixed P and a random x is $1/2^{\ell_1}$.

This fact discloses the trivial attack on the natural OPRF extension; for a fixed P after the OPRF protocol, the corrupt receiver can query to H_1 and find inputs satisfying $\text{Dcd}(P, x) = H_1(x)$. As the computational security parameter κ is much larger than those choices of ℓ_1 , the receiver can obtain q/ℓ_1 extra evaluations with q local computation of H_1 .

4.2 Malicious Flaw: Hardness of OKVS Overfitting Game

The OPRF (PSI) protocol of PRTY [9] takes ℓ_1 so that $\Pr[\mathcal{E}]$ in the PRTY bound (Lemma 1) is bounded by $2^{-\lambda}$, given $n' = c \cdot m$. One can check that larger n' yields smaller ℓ_1 , and such smaller ℓ_1 naturally brings better efficiency. Indeed, [9] set $n' \approx 12n$ to have efficient protocol where they reported experimental results. Meanwhile, Rindal and Schoppmann [13] (implicitly) argued that n' can be limited by only $m = (1 + \varepsilon_{\text{okvs}})n$, by setting $\ell_1 = \kappa$. This obviously makes it possible to take smaller ℓ_1 than [9] protocol, so that has better parameter choice and better performance of the OPRF protocol.¹ This has also been continuously adapted in the following works [3, 10]; whose main points are improving OKVS performance while following the framework of [13].

¹ We are not saying that every performance gain of [13] comes from such small ℓ_1 ; indeed the primary change from [9] to [13] that greatly affects to the performance is replacing OOS functionality into LPN-based VOLE.

However, we claim the choice of $\ell_1 = \kappa$ of RS construction with $n' = m$ is flawed. The reason is simple: $\ell_1 = \kappa$ is less than ℓ_1 derived from the PRTY bound with $n' = m$. That is, the malicious receiver with unbounded power can indeed find (more than) n' items x fitting in some P . However, such presence of unbounded adversary could be not considered as serious one, because one can simply assume the *computational* hardness of OKVS overfitting problem to make [13] secure. Indeed, Garimella et al. [7] stated that OKVS overfitting could be *computationally* hard even ℓ_1 is below PRTY bound.

What we further make is the concrete attack that shows OKVS overfitting can be computationally done, so that $\ell_1 = \kappa$ choice of [3, 10, 13] allows more PRF evaluations than $n' = m$ using less than 2^κ computational cost. The details are placed in the next section independently.

Flaw in the Security Proof. The Hybrid 4 of Lemma 2 in [13] is the relevant argument that argues n' can be limited to $m = (1 + \varepsilon_{\text{okvs}})n$. However, their argument only considered the case where the malicious adversary fixes the OKVS encoding P and then expects that $\text{Dcd}(P, x) = H_1(x)$ for additional x . This makes the proof false, because the malicious adversary can make $q = O(2^\kappa)$ amount of H_1 queries in advance, and then tries to find some maximal subset X' that overfits to some P of size m among the H_1 queries.

4.3 Efficacy of Extra Evaluation

In the OPRF view, extra evaluation directly violates the security requirements. However, the implication on PSI need some consideration. If the receiver obtains an additional PRF evaluation $F(y)$ for $y \in Y \setminus X$, this immediately violates the PSI functionality since the receiver knows other information than the intersection $X \cap Y$. The case where the additional PRF evaluation $F(y)$ is for $y \notin X \cup Y$ lets the receiver know that the element y is *not in* the sender's set Y , whose effect seems a bit ambiguous.

In fact, we can measure the amount of information leakage from the information $y \notin \bar{Y}$. Suppose that \mathcal{D}_1 be the original distribution of the sender's dataset \bar{Y} with $p_1(\bar{Y}) = \Pr_{Y \sim \mathcal{D}_1}[Y = \bar{Y}]$. And, let \mathcal{D}_2 be the distribution of the dataset without an element z in a universe of items in set. From the additional leakage of $z \notin \bar{Y}$, the corrupt receiver now have the distribution \mathcal{D}_2 of the sender's dataset. And the probability function of \mathcal{D}_2 can be computed as follows:

$$p_2(\bar{X}) = \begin{cases} \frac{1}{1-p'} \cdot p_1(\bar{X}) & \text{if } z \notin \bar{X} \\ 0 & \text{otherwise} \end{cases}$$

where $p' = \sum_{\{\bar{X} | z \in \bar{X}\}} p_1(\bar{X})$. Note that the probability $p_2(\bar{X})$ is increased proportionally to have sum of all the probability 1 when $z \notin \bar{X}$. Now we can compute the statistical distance between \mathcal{D}_1 and \mathcal{D}_2 ,

$$\text{dist}(\mathcal{D}_1, \mathcal{D}_2) = \sum_{z \notin \bar{X}} |p_1(\bar{X}) - p_2(\bar{X})| + \sum_{z \in \bar{X}} |p_1(\bar{X}) - p_2(\bar{X})| = 2p'.$$

From this, we conclude that the information of $y \notin Y$ can lead to a non-negligible amount of information, if y has a non-negligible probability ($> 2^{-\lambda}$) to be included in the honest party’s dataset. Moreover, this series of computations shows an undesirable feature of the PSI protocol, which is the dependency of the security on the distribution of the dataset.

5 Overfitting Attacks on Various OKVS

The ℓ_1 bound in [13] was found out to be information-theoretically not secure. To connect the information-theoretic vulnerability to a concrete attack, we need to overfit the OKVS P . Overfitting P refers to a problem to find $n' > m$ items $x_1, \dots, x_{n'}$ and a vector P that satisfy $\text{Dcd}(P, x_i) = H_1(x_i)$ for all i . Since the OKVS algorithm used in [13] is binary linear, there is a function $\text{row} : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_1}$ such that $\text{Dcd}(P, x) = \langle \text{row}(x), P \rangle$. Then overfitting P is in principle a finding $\vec{X} = (x_1, \dots, x_{n'})$ which builds a redundant (i.e., linearly dependent) system of linear equations $\text{row}(\vec{X}) \cdot P = H_1(\vec{X})$ for $n' > n$, where $\text{row}(\vec{X})$ (and $H_1(\vec{X})$) is a matrix whose i -th row is $\text{row}(x_i)$ (and $H_1(x_i)$).

For the idea of the attacks, observe that if there are some elements x_1, \dots, x_k in \vec{X} satisfy that $\sum_{j=1}^k \text{row}(x_j) \parallel H_1(x_j) = 0$, they contribute to the rank of $\text{row}(\vec{X})$ by at most $k - 1$. Finding such x_1, \dots, x_k is equivalent to solving the k -XOR problem for $k \geq 2$, and hence we can build \vec{X} of length $k(m-1)/(k-1) > m$ by solving the k -XOR problem repeatedly. Although $\text{row}(\vec{X})$ is a quite large matrix of size $k \times m$ where k -XOR problem is difficult in general, the attacks can be practically feasible thanks to the special structure of $\text{row}(\vec{X})$ observed in previous OKVS constructions [3, 10, 13], whose details are described in the following subsections.

Before presenting the attacks, we provide some definitions of basic problems required to formulate the attacks.

k -XOR Problem and Multi-collision Finding Problem. Before explaining our attacks, we briefly introduce the k -XOR problem and the multi-collision finding problem since it is a crucial subroutine of our attacks to solve these problems. The k -XOR problem is to find a k -tuple in some lists whose sum is 0. Formally, a k -XOR problem is defined as follows.

Definition 4 (k -XOR Problem). *Given lists of n -bit strings L_1, \dots, L_k , find k distinct elements $a_1 \in L_1, \dots, a_k \in L_k$ such that their XOR sum is zero:*

$$a_1 \oplus a_2 \oplus \dots \oplus a_k = 0.$$

The state of the art algorithm to solve the k -XOR problem is Wagner’s k -tree algorithm [17]. It costs $O(k \cdot 2^{n/(1+\lceil \log k \rceil)})$ time and space with lists of size $O(2^{n/(1+\lceil \log k \rceil)})$ each.

For a given cryptographic hash function, the multi-collision finding problem is to find c multi-collisions which give the same hash output. Formally, a multi-collision finding problem is defined as follows.

Definition 5 (*c*-Multicollision Finding Problem). *Given a hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$, find c distinct elements a_1, \dots, a_c such that:*

$$H(a_1) = H(a_2) = \dots = H(a_c).$$

The best algorithm to solve the c -multicollision finding problem is proposed by Suzuki et al. [15], which costs $O((c!)^{1/c} \cdot 2^{(c-1)n/c})$ time and space.

5.1 Overfitting PaXoS

PaXoS [9] is the firstly proposed OKVS with linear complexity. Given a vector of input items \vec{X} , which is an ordering of the set X , PaXoS has a following type of the $\text{row}(\vec{X})$ matrix.

$$\text{row}(\vec{X}) = [R_{\vec{X}} \mid D_{\vec{X}}]$$

where $R_{\vec{X}} \in \mathbb{F}_2^{n \times m}$ is defined by a pair of hash functions (h_1, h_2) each to $[m] = \{1, \dots, m\}$, and $D_{\vec{X}} \in \mathbb{F}_2^{n \times d}$ is a matrix with uniformly random entries. Each row of $R_{\vec{X}}$ corresponds to an item x , and it has only two 1's whose positions are $h_1(x)$ and $h_2(x)$. In the same row, the row of $D_{\vec{X}}$ also corresponds to the same item x , and it is defined by a uniform hash function $h_0 : \{0, 1\}^* \rightarrow \{0, 1\}^d$. We note that m is set to be $2.4n$ in [13].

As $R_{\vec{X}}$ is a sparse matrix, it is hard to apply Wagner's k -tree algorithm directly on $\text{row}(x) \parallel H_1(x)$. So, we will solve the k -XOR problem over $h_0(x) \parallel H_1(x)$ rather than $\text{row}(\vec{X}) \parallel H_1(x)$. To make $R_{\vec{X}}$ part also sum to zero, we bucketize $\text{row}(\vec{X}) \parallel H_1(x)$ by h_1 and h_2 , and deliberately organize a singular combination of buckets. For an easy example, if three items x_1, x_2, x_3 satisfy

$$\begin{aligned} h_1(x_1) &= 1, h_2(x_1) = 2, \\ h_1(x_2) &= 2, h_2(x_2) = 3, \\ h_1(x_3) &= 3, h_2(x_3) = 1, \end{aligned}$$

then $R_{\vec{X}}$ part of those three items sum to zero. Then, solving 3-XOR problem in those three buckets gives a small redundant system of equations of rank 2. Our attack on [13] basically repeats these steps until $\text{row}(\vec{X})$ has full rank, but the number of buckets may vary. In the following, we describe the detailed procedure of the attack.

- (Case $n' < 2(m-1)$) For this case, we reduce the $(n, \frac{k(m-1)}{k-1})$ -overfitting game to a k -XOR problem as follows. We will assume that $k-1 \mid m-1$, but the attack also works well otherwise.
 1. Let $Q = \{x_1, \dots, x_q\}$ be an arbitrary subset in $\{0, 1\}^*$. Query all the items in Q to h_0, h_1, h_2 , and H_1 . Bucketize Q by $\{h_1, h_2\}$, denoting B_{h_1, h_2} the corresponding bucket. Then, there are $\binom{m}{2}$ buckets, and there are expectedly $q/\binom{m}{2}$ items per bucket.
 2. Make a $k \times k$ -matrix K over \mathbb{F}_2 such that
 - $\text{rank}(K) = k-1$;

- $\text{rank}(K') = k - 1$ where $K' \in \mathbb{F}_2^{(k-1) \times (k-1)}$ is sub-matrix of K with first $k - 1$ rows and first $k - 1$ columns;
- each row of K should have only two 1's.

See Fig. 6 for examples.

3. Set $X' \leftarrow \emptyset$, and do the following for $j \in \{1, 1+(k-1), 1+2(k-1), \dots, m-k+2\}$.
 - (a) Denote the positions of two 1's in the i -th row ($1 \leq i \leq k$) of K by p_i and p'_i . Solve a k -XOR problem for $h_0(x) \parallel H_1(x)$ in buckets $B_{j+p_1, j+p'_1}, \dots, B_{j+p_k, j+p'_k}$.
 - (b) Denote the solution (x_1, \dots, x_k) , then it satisfies

$$\sum_{i=1}^k h_0(x_i) \parallel H_1(x_i) = 0.$$

Set $X' \leftarrow X' \cup \{x_1, \dots, x_k\}$

4. Let $\vec{X}' = (x'_1, \dots, x'_{n'})$ be an ordering of X' where $n' = \frac{k(m-1)}{k-1}$. Since $\text{rank}(\text{row}(\vec{X}')) = \text{rank}(\text{row}(\vec{X}') \parallel H_1(\vec{X}'))$, there is a solution P' of the linear equation $\text{row}(\vec{X}') \cdot P' = H_1(\vec{X}')$ where $H_1(\vec{X}') = (H_1(x'_1), \dots, H_1(x'_{n'}))$.

As solving a k -XOR problem of $(d + \ell_1)$ -bit strings costs $O\left(k2^{\frac{d+\ell_1}{1+\lceil \log k \rceil}}\right)$ time, our attack costs $O\left(m^2 2^{\frac{d+\ell_1}{1+\lceil \log k \rceil}}\right)$ time including time for querying to the random oracle. Note that the time cost for random oracle query dominates the cost for solving k -XOR problems.

- (Case $n' \geq 2m$) For this case, we reduce the $(n, c(m - 1))$ -overfitting game to a c -multicollision ($c \geq 2$) finding problem as follows.
 1. Let $Q = \{x_1, \dots, x_q\}$ be an arbitrary subset in $\{0, 1\}^*$. Query all the items in Q to h_0, h_1, h_2 , and H_1 . Bucketize Q by $\{h_1, h_2\}$, denoting B_{h_1, h_2} the corresponding bucket. Then, there are $\binom{m}{2}$ buckets, and there are expectedly $q/\binom{m}{2}$ items per bucket.
 2. Set $X' \leftarrow \emptyset$ and do the following for $j \in \{1, 2, \dots, m - 1\}$.
 - (a) Find a c -multicollision for $h_0(x) \parallel H_1(x)$ from bucket $B_{j, j+1}$.
 - (b) Gather the solutions of the problem to X' .
 3. Let $\vec{X}' = (x'_1, \dots, x'_{n'})$ be an ordering of X' where $n' = c(m - 1)$. Since $\text{rank}(\text{row}(\vec{X}')) = \text{rank}(\text{row}(\vec{X}') \parallel H_1(\vec{X}'))$, there is a solution P' of the linear equation $\text{row}(\vec{X}') \cdot P' = H_1(\vec{X}')$.

As finding c -multicollision of $(d + \ell_1)$ -bit strings costs $O((c!)^{1/c} 2^{\frac{(c-1)(d+\ell_1)}{c}})$ time, our attack costs $O(m^2 2^{\frac{(c-1)(d+\ell_1)}{c}})$ time for small enough c .

5.2 Overfitting 3H-GCT or RR22

Garimella et al. [7] proposed a 3H-GCT OKVS that generalizes PaXoS by using 3 hashes instead of 2 hashes. Raghuraman and Rindal [10] proposed a similar

$$\begin{array}{cc}
 \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \\
 \text{(a) } k = 4 & \text{(b) } k = 5
 \end{array}$$

Fig. 6. Examples of Step 2.

OKVS that has only w 1's in $R_{\bar{X}}$ part, which is almost the same with 3H-GCT for $w = 3$. Precisely, it also follows the form $\text{row}(\bar{X}) = [R_{\bar{X}} | D_{\bar{X}}]$, where $R_{\bar{X}} \in \mathbb{F}_2^{n \times m}$ is defined by a triple of hash functions (h_1, h_2, h_3) each to $[m]$, and $D_{\bar{X}} \in \mathbb{F}_2^{n \times d}$ is a matrix with uniformly random entries; 3H-GCT proposal sets $\ell_1 = 1$, and RR22 sets $\ell_1 = \kappa$. We denote $d = d' \ell_1$. Each row of $R_{\bar{X}}$ corresponds to an item x , and it has only three 1's whose positions are $h_1(x)$, $h_2(x)$ and $h_3(x)$. Each row of $D_{\bar{X}}$ also corresponds to the same item x , and it is defined by a uniform hash function $h_0 : \{0, 1\}^* \rightarrow \{0, 1\}^d$. We note that m is set to be approximately $1.3n$ in [13].

As PaXoS and 3H-GCT are similar, the attacks for PaXoS work well on this case except some details. The buckets should be labeled with three hash values $\{h_1, h_2, h_3\}$. In Step 2, since 3H-GCT uses three hash functions, the submatrix should be of the different form. Step 2 can be rephrased to attack 3H-GCT as follows.

2. For an even integer $k > 2$, make a $k \times (3k/2)$ -matrix over \mathbb{F}_2 of rank $k - 1$. Each row of this matrix should have only three 1's. See Fig. 7 for examples.

The asymptotic time complexity should be $O(m)$ times larger than that for PaXoS.

$$\begin{array}{cc}
 \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \\
 \text{(a) } k = 4 & \text{(b) } k = 6
 \end{array}$$

Fig. 7. Examples of submatrices when attacking 3H-GCT.

5.3 Overfitting RB-OKVS

RB-OKVS is an OKVS proposed by Bienstock et al. [3], whose $\text{row}(\vec{X})$ has no sparse part unlike PaXoS or 3H-GCT. Given a pair of hash functions $h_1 : \{0, 1\}^* \rightarrow [m - d]$ and $h_0 : \{0, 1\}^* \rightarrow \{0, 1\}^d$, $\text{row}(x)$ for an item x is defined by

$$\text{row}(x)[i] = \begin{cases} h_0(x)[i - h_1(x)] & \text{if } h_1(x) < i \leq h_1(x) + d \\ 0 & \text{otherwise} \end{cases}$$

where $v[n]$ for a vector v denotes the n -th component of v .

The attack can be done in the almost similar way to PaXoS and 3H-GCT. The details can be found in Appendix A due to the space limit.

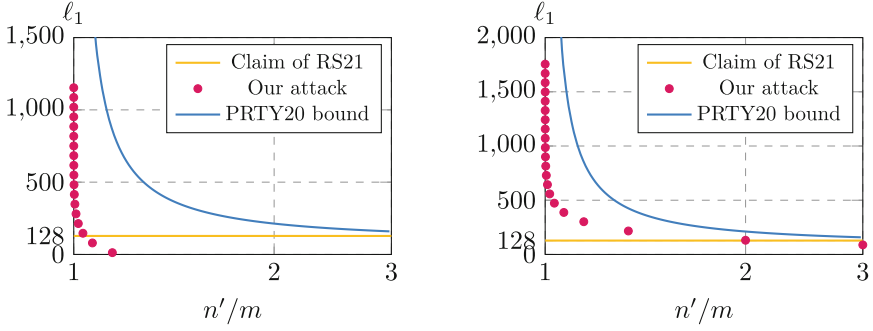
5.4 Efficacy of the Attacks

To measure the effectiveness of the attacks, we plot three dot graphs of ℓ_1 corresponding to the extra-evaluation ratio n'/m for each OKVS in Fig. 8b, 8a, and 8c. Common to all the three graphs, the blue curve is the PRTY bound [9] for each parameter set, and the yellow line is ℓ_1 which was claimed secure in [13]. We note that [3, 10] follow the choice of ℓ_1 . The red dots are the least ℓ_1 to prevent our attacks. Each red dot corresponds to either $3 \leq k \leq 20$ of the k -XOR problem or $2 \leq c \leq 3$ of the c -multicollision finding problem. The hash length ℓ_1 to prevent our attacks are much larger than ℓ_1 claimed in [13]. Each graph is plotted for $n = 2^{20}$, and the detailed choice of parameters is written in each figures.

When the attacks are applied in the context of PSI, it implies that a corrupt receiver can input n' random items rather than n chosen items. As data is not distributed uniformly at random in practice, n' random items seem less valuable than n chosen items so the attacks are quite useless. However, the attacks can still evaluate more items than claimed including n chosen items as follows.

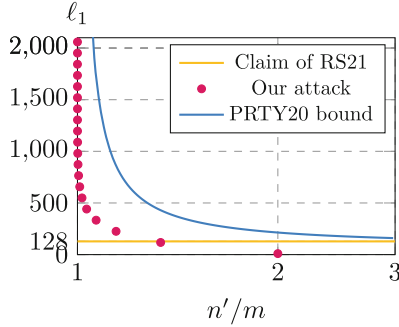
1. For a fixed (ordered) set $\vec{X} = (x_1, \dots, x_n)$, make the $\text{row}(\vec{X})$ matrix.
2. Using Gaussian elimination, find all the non-pivot positions. There are at least $(m - n)$ non-pivot positions.
3. Let Q be a set of items whose h_i ($1 \leq i \leq 3$) values are all in the non-pivot positions.
4. Mount the attack with Q as if there is no pivot positions.

It is straightforward that this variation for RS21 and RR22 can evaluate $n + \frac{n'}{m} \cdot (m - n)$ with similar complexities. For BPSY23, it is not always true. If a consecutive d positions do not include $k - 1$ non-pivot positions, the attack cannot utilize all the non-pivot positions so that the number of evaluation may be less than $\frac{n'}{m} \cdot (m - n)$. For the worst case (most of the consecutive d positions do not include $k - 1$ non-pivot positions), this variation may not evaluate more than m items.



(a) (RR22 [10]) We set the number of hashes 3, the expansion ratio 1.23, and the number of $\mathbb{F}_{2^{128}}$ -dense columns 2.

(b) (PaXoS [9]). We set the number of hashes 2, the expansion ratio 2.4, and the number of dense columns 40.



(c) (RB-OKVS [3]). We set the expansion ratio 1.1, and the number of dense columns 206.

Fig. 8. ℓ_1 to prevent the attack to various protocols.

6 Generic Security Considerations

In this section, we propose generic mitigations for our attacks, and a modified OKVS-based OPRF with provably secure choice of parameters.

6.1 Mitigations for Attacks and Revised Parameter Selection

For the semi-honest case, note that the attack in Sect. 4.1 allows one additional PRF evaluation with probability $1/2^{\ell_1}$. Thus, to prevent this attack, we recommend to raise ℓ_1 to at least κ for the semi-honest case, from the previous works choice $\ell_1 = \lambda + 2 \log n$.

For the malicious case, the adversary can try to overfit OKVS, which finds additional PRF evaluation more efficiently. Here, observe that the possibility of OKVS overfitting depends on the number of H_1 queries, say q , as well as the length ℓ_1 . If we can restrict q less than 2^κ , OKVS overfitting would get harder

and then we can utilize more efficient OPRF parameters. This can be simply done by set a sort of timeout for the (corrupt) receiver. To do this, we modify two parts of the protocol, as following.

1. At the beginning of the protocol, the sender samples random salt salt and transmit it to receiver. Then, both parties incorporates salt into all random oracle inputs. This clearly prevents the malicious receiver prepare H_1 values before the protocol starts.
2. The sender aborts the protocol if the time between sending the salt and receiving the correction matrix back from the receiver exceeds a certain amount of time.

Given that the overfitting attack is more effective than the semi-honest attack, this mitigation technique has an effect of reducing the number of queries for the overfitting attack. We formalize this problem as a new overfitting game as follows.

Definition 6 ($(n, n', q_{\text{on}}, q_{\text{off}})$ -OKVS online overfitting game). *Let (Ecd, Dcd) be an OKVS with parameters chosen to support n items, and let \mathcal{A} be an arbitrary PPT adversary. Run $P \leftarrow \mathcal{A}^{H_1}(1^\kappa, q_{\text{on}})$ where q_{on} is the number of queries to H_1 before outputting P . Define*

$$X' = \{x \mid \mathcal{A} \text{ queried } H_1 \text{ at } x \text{ and } \text{Dcd}(P, x) = H_1(x)\}$$

where \mathcal{A} can query to H_1 less than q_{off} times after P is produced. If $|X'| > n'$, then the adversary wins the $(n, n', q_{\text{on}}, q_{\text{off}})$ -OKVS online overfitting game.

The online query complexity q_{on} should be determined after reviewing a number of factors; such as, computing power of participating parties, timeout time, or network environments. Given that recent Bitcoin’s hash rate is about 2^{69} hashes per second,² if those factors are unknown beforehand, $q_{\text{on}} = 2^{96}$ seems a safe choice for not a long timeout time. The offline query complexity q_{off} is the original amount of permitted query, which is usually $O(2^\kappa)$.

Concrete Choice of ℓ_1 . We provide some concrete choices of ℓ_1 that makes OKVS online overfitting game. For that, we first consider n'_1 by the maximum number of allowed evaluations following the PRTY bound given that q_{on} random oracle queries are permitted. Then we let n'_2 be the least integer such that

$$\Pr_{S \leftarrow B(q_{\text{off}}, 2^{-\ell_1})} [S > n'_2] < 2^{-\lambda},$$

which can be rewritten by multiplicative Chernoff bound as follows.

$$\left(\frac{eq_{\text{off}}}{n'_2 2^{\ell_1}} \right)^{n'_2} < 2^{-\lambda}$$

Then, this ℓ_1 is a secure choice of $n' = n'_1 + n'_2$ allowed evaluations. Table 1 summarizes ℓ_1 according to n' and q_{on} .

² This data is retrieved from <https://www.blockchain.com/explorer/charts/hash-rate> in May 2024.

Table 1. The choice of ℓ_1 depending on q_{on} and n' , where m is set to be 1.3×2^{20} , and q_{off} is set to be 2^{128} . For $q_{\text{on}} = 2^{128}$, the offline complexity q_{off} is set to be 0, which is same as the original PRTY bound with $q = 2^{128}$.

$\log q_{\text{on}}$ \backslash n'/m	1.5	2	3	4	5
32	111	110	109	108	108
64	134	112	109	108	108
96	226	150	113	111	109
128	322	214	160	141	132

In Table 1, a reader might feel that ℓ_1 converges to 108 when n'/m is sufficiently large, whatever q_{on} is. It is because the semi-honest attack allows 2 times more evaluation if ℓ_1 is decreased by a single bit. So, ℓ_1 can be less than 108 if n'/m is large enough; if $n'/m = 1000$, then $\ell_1 = 100$.

On ℓ_1 for the Semi-honest Model. At the beginning of this section, we recommend to set $\ell_1 = \kappa = 128$ for the semi-honest model. Meanwhile, the choices of ℓ_1 for the malicious model presented in Table 1 for $q_{\text{on}} \leq 2^{96}$ is less than 128. One may think this weird, because the malicious model allows smaller ℓ_1 than the semi-honest model. However, we stress that not only the adversarial model that determines the possible attacks, but the bound n' for the number of PRF evaluations is also an important factor for ℓ_1 : In the semi-honest model, the bound n' is implicitly set by n , whereas the malicious model allows somewhat larger n' such as $c \cdot m$ for $m = (1 + \varepsilon_{\text{okvs}})n$. Finally, it would be possible to extend the definition of semi-honest OPRF to allowing further PRF evaluation n' , which is not explicitly done in our OPRF definition, which enables to set $\ell_1 = 128 - \log n'$ for the semi-honest model.

6.2 Revised Security Proof

We present the revised OKVS-based OPRF construction in Fig. 9, and Theorem 2 that specifies the correct conditions of parameters, which reflects the previous works the security flaws.

Correctness. We check that the set $\{H_2(\text{Dcd}(V, x) + w) \mid x \in X\}$ indeed equals to $F(X)$ with the sender's definition

$$F(x) = H_2(x, \text{Dcd}(W', x) + w - \vec{\Delta} \odot C(H_1(x, \text{salt}))).$$

For the readability, we omit `salt` in H_1 and w for a while, which are common for both parties. Then it holds that from the linearity of `Dcd` and linear code C

$$\begin{aligned}
\text{Dcd}(W', y) - \vec{\Delta} \odot C(H_1(y)) &= \text{Dcd}(V + \vec{\Delta} \odot C(P), y) - \vec{\Delta} \odot C(H_1(y)) \\
&= \text{Dcd}(V, y) - \vec{\Delta} \odot (\text{Dcd}(C(P), y) - C(H_1(y))) \\
&= \text{Dcd}(V, y) - \vec{\Delta} \odot C(\text{Dcd}(P, y)) - C(H_1(y)) \\
&= \text{Dcd}(V, y) - \vec{\Delta} \odot C(\text{Dcd}(P, y) - H_1(y)) \quad (2)
\end{aligned}$$

for any $y \in \{0, 1\}^*$. Note that $\text{Dcd}(C(P), y) = C(\text{Dcd}(P, y))$ requires that $\text{row}(y)$ is a vector of \mathbb{B}^{n_c} for every $y \in \{0, 1\}^*$, which is always true when the underlying OKVS is binary. Now, the OKVS correctness property ensures $\text{Dcd}(P, x) = H_1(x)$ for every $x \in X$, then Eq. (2) further becomes

$$\begin{aligned}
\text{Dcd}(W', x) - \vec{\Delta} \odot C(H_1(x)) &= \text{Dcd}(V, x) - \vec{\Delta} \odot C(\text{Dcd}(P, x) - H_1(x)) \\
&= \text{Dcd}(V, x) - \vec{\Delta} \odot C(H_1(x) - H_1(x)) \\
&= \text{Dcd}(V, x),
\end{aligned}$$

which concludes the correctness of our framework.

Security Proof. The underlying proof idea is similar to the previous works [9, 13], whose main idea is to show that $D_y := \text{Dcd}(P, y) - H_1(y)$ has sufficient ($\geq \kappa$) entropy. However, our statement and proof have two distinct points. First, it provides general conditions on the linear code $[n_c, k_c, d]_{\mathbb{B}}$ for arbitrary choice of \mathbb{F} and \mathbb{B} , which correctly applies to both previous constructions. Second, our proof correctly provides the condition on ℓ_1 considering our proposed attack; namely online OKVS overfitting game.

Theorem 2. *Let \mathbb{F} be a field, \mathbb{B} be a subfield of \mathbb{F} , and $\ell_1 > 0$ be an integer that makes $(n, n', q_{\text{on}}, q_{\text{off}})$ -online OKVS overfitting game hard. Let $C = [n_c, k_c, d]_{\mathbb{B}}$ be a \mathbb{B} -linear code where $k_c \cdot \log |\mathbb{B}| \geq \ell_1$ and $d \cdot \log |\mathbb{F}| \geq \kappa$. Then the protocol of Fig. 9 securely realizes $\mathcal{F}_{\text{opr}}^{\text{prf}}$ against $(q_{\text{on}}, q_{\text{off}})$ -malicious adversary in a $\mathcal{F}_{\text{vole}}(\mathbb{F}, \mathbb{B})$ -hybrid model.*

We prove the theorem by the following two lemmas.

Lemma 2. *The protocol of Fig. 9 securely realizes $\mathcal{F}_{\text{opr}}^{\text{prf}}$ against malicious sender \mathcal{A} who queries to random oracles at most q times.*

Proof. The \mathcal{S} interacts with \mathcal{A} as follows:

- \mathcal{S} plays the role of $\mathcal{F}_{\text{vole}}$. \mathcal{S} waits for \mathcal{A} to send $(\vec{W}_i, \Delta_i)_{i \in [n_c]} \in (\mathbb{F}^m \times \mathbb{F})^{n_c}$.
- On behalf of the receiver, \mathcal{S} waits for \mathcal{A} to send `salt` and c^s . Then, \mathcal{S} sends uniform $w^r \in \mathbb{F}^{n_c}$ and $U' \in \mathbb{B}^{m \times n_c}$. Next, \mathcal{S} waits for \mathcal{A} to send w^s and aborts if $c^s \neq H(w^s)$.
- For every queries to H , \mathcal{S} abort if there exists output collision.
- Whenever \mathcal{A} queries $H_2(y, q)$, if $q = \text{Dcd}(W', y) + w - \vec{\Delta} \odot C(H_1(y, \text{salt}))$ and $H_2(y, q)$ has not previously been queried, send y to $\mathcal{F}_{\text{opr}}^{\text{prf}}$ and programs H_2 to the response. Otherwise, H_2 responses normally.

Parameters: A finite field \mathbb{F} with a subfield \mathbb{B} , and a $[n_c, k_c, d]_{\mathbb{B}}$ -linear code C . The input set size n_x , and three random oracles $H : \{0, 1\}^* \rightarrow \{0, 1\}^{2\kappa}$, $H_1 : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_1}$ such that $\ell_1 \leq \log |\mathbb{B}| \cdot k_c$ and $H_2 : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_2}$. A linear OKVS algorithm pair (Ecd, Dcd) of expansion factor $\varepsilon_{\text{okvs}}$.

Protocol: Upon an input set X from the receiver, the protocol runs as follows:

1. Two parties execute n_c times of (\mathbb{F}, \mathbb{B}) -sVOLE of length $m = (1 + \varepsilon_{\text{okvs}}) \cdot n$. In each i -th execution, the sender obtains $\vec{W}^i \in \mathbb{F}^m$ and $\Delta_i \in \mathbb{F}$ and the receiver obtains $\vec{V}^i \in \mathbb{F}^m$ and $\vec{U}^i \in \mathbb{B}^m$. Two parties let $W, V \in \mathbb{F}^{m \times n_c}$ ($U \in \mathbb{B}^{m \times n_c}$, resp) as matrices of i -th column vector \vec{W}^i, \vec{V}^i (\vec{U}^i , resp).
2. The sender samples $\text{salt} \leftarrow \{0, 1\}^\kappa$, and $w^s \leftarrow \mathbb{F}^{n_c}$, and compute $c^s = H_1(w^s)$. Then it sends salt and c^s to the receiver.
3. The receiver samples $w^r \leftarrow \mathbb{F}^{n_c}$, and computes an OKVS encoding $P \in \{0, 1\}^{m \times \ell_1}$ on the set $\{(x, v_x) : x \in X, v_x = H_1(x, \text{salt})\}$.
4. The receiver applies C on each row vector $\vec{P}_i \in \{0, 1\}^{\ell_1}$ of P by embedding it into \mathbb{B}^{k_c} . Write the resulting matrix by $C(P) \in \mathbb{B}^{m \times n_c}$ whose i -th row is $C(\vec{P}_i) \in \mathbb{B}^{n_c}$.
5. The receiver sends w^r and the correction matrix $U' = C(P) - U \in \mathbb{B}^{m \times n_c}$ to the sender who locally computes $W' = W + \vec{\Delta} \odot U'$.
6. The sender sends w^s to the receiver, who aborts if $c^s \neq H_1(w^s)$. Two parties define $w := w^s + w^r$.
7. The receiver outputs $\{H_2(x, \text{Dcd}(V, x) + w) \mid x \in X\}$ (as $F(X)$).
8. The sender defines the OPRF function $F : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_2}$ by

$$F(y) := H_2 \left(y, \text{Dcd}(W', y) + w - \vec{\Delta} \odot C(H_1(y, \text{salt})) \right).$$

Fig. 9. Modified OKVS-based OPRF construction.

To prove that this simulation is indistinguishable, consider the following hybrids.

- G_0 : The same as the real protocol except \mathcal{S} plays the role of $\mathcal{F}_{\text{vole}}$.
- G_1 : \mathcal{S} aborts if there exists output collision in H queries. As H as output length 2κ ,

$$\Pr[G_1 \text{ aborts}] \leq \frac{q^2}{2^{2\kappa}}$$

- G_2 : \mathcal{S} samples U' uniformly instead of computing $U' = C(P) - U$. Since U is chosen uniformly at random, the distribution of \mathcal{A} 's view does not change except when Ecd aborts. Since the probability that Ecd abort is less than $2^{-\lambda}$, the difference from the previous game is negligible:

$$|\Pr[\mathcal{A} \text{ wins } G_1] - \Pr[\mathcal{A} \text{ wins } G_2]| \leq \Pr[\text{Ecd aborts in } G_0] \leq \frac{1}{2^\lambda}$$

- G_3 : When \mathcal{S} samples U' and w^r , abort if there exists (y, σ) such that $H_2(y, \sigma)$ has previously been queried and $\sigma = \text{Dcd}(W', y) + w - \vec{\Delta} \odot C(H_1(y))$. As w^r is chosen uniformly at random, we have

$$\Pr[G_3 \text{ aborts.}] \leq \frac{q}{|\mathbb{F}|^{n_c}} \leq \frac{q}{2^\kappa}$$

Note this hybrid represents why w^r should be sampled by the receiver and sent to the sender.

- G_3 : Whenever \mathcal{A} queries $H_2(y, \sigma)$ after U' is sampled, if $\sigma = \text{Dcd}(W', y) + w - \vec{\Delta} \odot C(H_1(y))$ and $H_2(y, \sigma)$ has not previously been queried, send y to $\mathcal{F}_{\text{oprf}}$ and programs H_2 to the response. Otherwise, H_2 responses normally. As output distribution of $\mathcal{F}_{\text{oprf}}$ is random, the distribution of \mathcal{A} 's view is identical. \square

Lemma 3. *The protocol of Fig. 9 securely realizes $\mathcal{F}_{\text{oprf}}$ against $(q_{\text{on}}, q_{\text{off}})$ -malicious receiver \mathcal{A} .*

Proof. The \mathcal{S} interacts with \mathcal{A} as follows:

- \mathcal{S} plays the role of $\mathcal{F}_{\text{vole}}$. \mathcal{S} waits for \mathcal{A} to send $(\vec{V}_i, \vec{U}_i)_{i \in [n_c]} \in (\mathbb{F}^m \times \mathbb{B}^m)^{n_c}$.
- On behalf of sender, \mathcal{S} sends uniform salt and c^s to \mathcal{A} .
- When \mathcal{A} sends w^r , sample uniform w^s and program $H(w^s) = c^s$.
- When \mathcal{A} sends U' , compute $U' - U = C(P)$. For $H_1(x)$ queries made by \mathcal{A} , \mathcal{S} checks if $\text{Dcd}(C(P), x) = C(H_1(x, \text{salt}))$, \mathcal{S} sends x to $\mathcal{F}_{\text{oprf}}$ and receives it as $F(x)$.
- For each $x \in X$, \mathcal{S} programs $H_2(x, \text{Dcd}(W', x) + w - \vec{\Delta} \odot C(H_1(x)))$ as $F(x)$.

To prove that this simulation is indistinguishable, consider the following hybrids.

- G_0 : The same as the real protocol except \mathcal{S} plays the role of $\mathcal{F}_{\text{vole}}$, so \mathcal{S} waits for \mathcal{A} to send $(\vec{V}_i, \vec{U}_i)_{i \in [n_c]} \in (\mathbb{F}^m \times \mathbb{B}^m)^{n_c}$.
- G_1 : When \mathcal{S} samples uniform salt, \mathcal{S} abort if there exists a prior query from \mathcal{A} to H_1 with form $H_1(\cdot, \text{salt})$. As $|\text{salt}| \geq \kappa$, we have

$$\Pr[G_1 \text{ aborts}] \leq \frac{q_{\text{off}}}{2^\kappa}$$

- G_2 : When \mathcal{A} sends U' and w^r , compute $U' - U = C(P)$. For each of the previous $H_1(x, \text{salt})$ queries made by \mathcal{A} , \mathcal{S} checks if $\text{Dcd}(C(P), x) = C(H_1(x, \text{salt}))$ and if so adds x to X . \mathcal{S} sends X to $\mathcal{F}_{\text{oprf}}$ and receives $\{F(x) \mid x \in X\}$ in response.
- G_3 : \mathcal{S} samples uniform c^s instead of computing H , but programs $H(w^s) = c^s$ after sampling w^s uniformly at random. G_3 aborts if there exists a previous query to H with input collision or there exists a previous query $H_2(y, \sigma)$ such that $\sigma = \text{Dcd}(V, y) + w$. Then, as w^s is chosen uniformly at random from \mathbb{F}^{n_c}

$$\Pr[G_2 \text{ aborts}] \leq \frac{q_{\text{off}}}{|\mathbb{F}|^{n_c}} \leq \frac{q_{\text{off}}}{2^\kappa}$$

Next, \mathcal{S} programs $H_2(x, \text{Dcd}(V, x) + w) = F(x)$ for $x \in X$ and by the uniformity of $F(x)$, it does not change \mathcal{A} 's view. Note that this hybrid represents why w^s is required.

- G_4 : For each query $H_2(x, \sigma)$, \mathcal{S} add x into X and programs $H_2(x, \sigma) = \mathcal{F}_{\text{opr}}(x)$ if

$$\begin{cases} \sigma = \text{Dcd}(V, x) + w \\ \text{Dcd}(C(P), x) = C(H_1(x, \text{salt})) \end{cases}$$

then, add x to X and program $H_2(x, \sigma) = \mathcal{F}_{\text{opr}}(x)$. Similarly, for each query $H_1(x, \text{salt})$, \mathcal{S} add x into X and programs $H_2(x, \text{Dcd}(V, x) + w) = \mathcal{F}_{\text{opr}}(x)$ if

$$\text{Dcd}(C(P), x) = C(H_1(x, \text{salt})).$$

By the uniformity of $\mathcal{F}_{\text{opr}}(x)$, this does not change the view of \mathcal{A} . Note that $|X| \leq n'$, by the hardness of $(n, n', q_{\text{on}}, q_{\text{off}})$ -online OKVS overfitting problem.

- G_5 : At the end of protocol, \mathcal{S} samples Δ and aborts if \mathcal{A} ever makes an $H_2(x, \sigma)$ query for $x \notin X$ such that

$$\sigma = \text{Dcd}(W', x) - \vec{\Delta} \odot C(H_1(x, \text{salt})).$$

By (2), for queries $H_2(x, \sigma)$ such that $\text{Dcd}(P, x) \neq H_1(x, \text{salt})$, as there is at least $d \log |\mathbb{F}|$ -bit entropy from $\vec{\Delta}$, we have

$$\Pr[G_5 \text{ abort}] \leq \frac{Q}{2^\kappa}.$$

□

6.3 Double Execution of OPRF

This section proposes a novel idea for the extreme case where much tighter $n' \approx m$ is required. Recall that the underlying idea of preventing OKVS overfitting is to limit the number of H_1 queries that the malicious receiver can obtain. The basic idea of what we call double execution of OPRF is to change H_1 by another OPRF, say F , and using the value again to encode OKVS. Precisely, for a given OPRF functionality OPRF_n for n items with variable number of allowed evaluations, the double execution proceeds as follows.

1. The sender and the receiver invoke OPRF_n with $n'' \gg m$ allowed evaluations. The receiver gets at most n'' evaluated items $\{F(x_1), \dots, F(x_{n''})\}$. We will denote the bit-length of H_1 in this phase ℓ''_1 .
2. The sender and the receiver invoke OPRF_n one more time. But the value set V_X at OKVS encoding step at receiver's side becomes $\{F(x) : x \in X\}$ instead of $\{H_1(x) : x \in X\}$. As a result, the receiver gets at most n' evaluated items $\{G(x_{i_1}), \dots, G(x_{i_{n'}})\}$ where $i_k \in [n']$.

The double execution has in fact the same effect of constraining online complexity. In the second phase, since the receiver use $F(x)$ instead of $H_1(x)$, a corrupt receiver can try to overfit an OKVS with only n'' random oracle queries. As ℓ_1 required for tight $n' \approx m$ is so huge in the PRTY bound, the double execution is an economic choice for a tight $n' \approx m$. For example, if a sender and a receiver want to achieve $n' = 1.02m$ in [10] with $q_{\text{on}} = 2^{128}$, $n = 2^{20}$, ℓ_1 should be no less than 5492 bits by PRTY bound. If they use double execution in this case, it is sufficient that $\ell''_1 = 109$ for $n'' = 24m$ and $\ell_1 = 307$ for $n' = 1.02m$. The communication of double execution is $13.2 \approx 5492/(109 + 307)$ times smaller than the single execution.

7 Better OPRFs from Intermediate Fields

Although the original purpose of Fig. 9 was to address both the PRTY construction (with $\mathbb{F} = \text{GF}(2)$) and the RS construction (with $\mathbb{F} = \text{GF}(2^{128})$) simultaneously, it also demonstrates another instantiation with $\mathbb{F} = \text{GF}(2^f)$ for $1 < f < \kappa$, beyond the binary field $\text{GF}(2)$ and the full κ -degree field $\text{GF}(2^\kappa)$. This section discusses these alternative instantiations and reveals more efficient OKVS-based OPRF protocols compared to previous ones.

7.1 Concrete Instantiations

We first consider the instantiation with small-sized \mathbb{F} , for example, $\mathbb{F} = \text{GF}(2^f)$ with $f \leq 10$. In this case, the underlying VOLE can be efficiently realized by the protocol due to [14], referred to as **SoftSpokenVOLE**, which is based solely on Minicrypt assumptions. The computation cost of the VOLE scheme grows exponentially with respect to the parameter f , limiting its practical application when f becomes too large. As a result, the experiments conducted in [14] were restricted to values of f up to 10.

For larger fields \mathbb{F} , the LPN-based VOLE protocol, known as **SilentVOLE** [5], offers a more efficient solution compared to the **SoftSpokenVOLE** protocol. It has a linear computational complexity with respect to the field size, making it suitable for handling large fields without incurring excessive computational costs. However, due to the linear complexity of the VOLE, there is no benefit to using $n_c > 1$. Therefore, we only consider small-sized \mathbb{F} hereafter.

Choice of \mathbb{B} . To complete the instantiation, we need to specify the choice of the subfield degree b that determines $\mathbb{B} = \text{GF}(2^b)$, which can be freely chosen among the divisors of f . Here, we highlight two advantages of taking $b = 1$.

First, the main computation of **SoftSpokenVOLE** consists of \mathbb{B} operations, so taking $b = 1$ makes every computation in the VOLE as bit-operations, resulting in the fastest performance. In fact, the original proposal and its application [2] used $b = 1$ for efficiency. Second, the transmission of the correction matrix $U' \in \mathbb{B}^{m \times n_c}$ (from the receiver to the sender) dominates the communication cost of our OPRF protocol with $m \cdot n_c \cdot \log |\mathbb{B}| = m \cdot n_c \cdot b$ bits. As m is determined by the input set size n and OKVS expansion factor ε_{okvs} , the choice of b only affects $n_c \cdot b$. Considering the general bound $n_c \geq d_c + k_c - 1$ for every linear code and the conditions $k_c \geq \ell_1/b$ and $d_c \geq \kappa/f$ in Theorem 2, we have the inequality $n_c \cdot b \geq \kappa \cdot b/f + \ell_1 - b$. This implies that $b = 1$ allows the minimal communication cost. However, for $b = 1$, the linear code achieving $n_c = d_c + k_c - 1$ (called maximal distance separable code) does not always exist, and that argument cannot assure that $b = 1$ is the best choice. For these reasons, we investigate the known lower bounds of n_c such that the linear code $[n_c, k_c, d_c]_{\mathbb{B}}$ exists in [1] for given k_c and d_c in Table 2, which shows that $b = 1$ provides the minimal $n_c \cdot b$.

Meanwhile, the choice of b affects other computational parts. The computation of $F(\cdot)$, particularly the OKVS decoding $\text{Dcd}(W', x)$ or $\text{Dcd}(V, x)$ where

Table 2. Some minimal possible values of n_c and corresponding $n_c \cdot b$ where $[n_c, k_c, d_c]_{\mathbb{B}}$ can exist. The length $\ell_1 = 109$ (resp., 150) is taken from Table 1 with $q_{\text{on}} = 2^{96}$ with $n' = 5m$ (resp., $2m$).

$\ell_1 = 109$						$\ell_1 = 150$					
\mathbb{F}	d_c	\mathbb{B}	k_c	n_c	$n_c \cdot b$	\mathbb{F}	d_c	\mathbb{B}	k_c	n_c	$n_c \cdot b$
		GF(2)	109	149	149			GF(2)	150	192	192
GF(2^8)	16	GF(2^2)	55	78	156	GF(2^8)	16	GF(2^2)	75	99	198
		GF(2^8)	14	29	232			GF(2^8)	19	34	272
GF(2^6)	22	GF(2)	109	162	162	GF(2^6)	22	GF(2)	150	206	206
		GF(2^2)	55	86	172			GF(2^2)	75	107	214
		GF(2^3)	37	62	186			GF(2^3)	50	75	225
		GF(2^6)	19	40	240			GF(2^6)	25	46	276
GF(2^4)	32	GF(2)	109	184	184	GF(2^4)	32	GF(2)	150	230	230
		GF(2^2)	55	99	198			GF(2^2)	75	120	240
		GF(2^4)	28	59	236			GF(2^4)	38	69	276
GF(2^2)	64	GF(2)	109	250	250						
		GF(2^2)	55	145	290						

$W', V \in \mathbb{F}^{m \times n_c}$, becomes maximal when $b = 1$ since it consists of XOR operations of $(n_c \cdot f)$ -bit strings. Furthermore, two parties need to execute n_c times of (\mathbb{F}, \mathbb{B}) -VOLE, and the number of VOLE instances would be maximal for $b = 1$.

Considering these facts, another choice of b other than 1 could provide a trade-off between computation and communication costs. As verifying the computational benefit of other choices of b requires another extensive experimental effort, we choose to fix $b = 1$ for its clear communication advantage and leave the investigation of further trade-offs with other choices of $b \neq 1$ for future work.

Communication Cost. This framework consists of two phases of interaction: VOLE, and the transmission of $U' = C(P) - U \in \mathbb{B}^{m \times n_c}$ from the receiver to the sender. Assuming the VOLE communication is negligible (a reasonable assumption due to recent advances in VOLE), the second phase dominates the total communication cost, specifically

$$\text{comm}_{\text{oprf}} = (n_c \log |\mathbb{B}|) \cdot m = (n_c \log |\mathbb{B}|) \cdot (1 + \varepsilon_{\text{okvs}}) \cdot n. \quad (3)$$

Comparison with RS Constructions. As the RS construction that combines LPN-based VOLE and OKVS represents the state-of-the-art for OPRF and PSI, we provide some comparisons with it. To recall, the RS construction can be understood as Fig. 9 with $\ell_1 = 128$, $\mathbb{F} = \mathbb{B} = \text{GF}(2^{128})$ and the identity linear map $[1, 1, 1]_{\mathbb{B}}$, which requires $128 \cdot m$ bits of communication, as shown in Eq. (3). As the value of $n_c \cdot b$ in Table 2 always exceeds 128, the RS construction still incurs a smaller communication cost. However, another important factor is the bound for PRF evaluation n' . The RS construction itself does not have the online

hash mitigation, and the PRF bound n' should be calculated with respect to 2^{128} queries, resulting in $n' = 5.6m$ with $m = 1.3n$ (assuming OKVS from [10]). On the other hand, our choice of $\ell_1 = 109$ and 150 comes from $n' = 5m$ and $2m$, respectively, which is smaller than the RS construction's $n' = 5.6m$. In other words, for the original RS construction to achieve $n' = 5m$ or $2m$, their ℓ_1 should be taken larger.

7.2 Performance Evaluation

We present some experimental results to evaluate performance of our newly proposed protocol. Since there is a trade-off between computational and communication complexities, our evaluation was carried out under different network settings with varying the field degree f from 1 to 8 (while fixing the subfield degree by $b = 1$), and the set size $n = 2^{20}$.

For the SoftSpokenVOLE realization, we utilize the implementation of libOTe library [12]. We also choose to employ the OKVS algorithm proposed by [10], whose implementation is publicly available at [16]. Note that this the okvs expansion factor $\varepsilon_{\text{okvs}} \approx 0.3$, so every m below can be regarded as $1.3n$. We remark that RB-OKVS [3] is claimed to have smaller $\varepsilon_{\text{okvs}}$ while having similar performance with [10]. However, we found no public implementation of RB-OKVS, and our internal implementation of RB-OKVS cannot reproduce the numbers in the original paper, so we simply use [10] for experiments. Note that the choice of OKVS might change the absolute numbers in this section, but our main contents are almost independent to the choice of OKVS.

The tests were performed using a machine equipped with 3.50 GHz Intel Xeon E5-1650 v3 (Haswell) CPU and 128 GB RAM, using a single thread. For concrete parameters, we use $\lambda = 40$ bit of statistical security and $\kappa = 128$ bits of computational security. To simulate various network environments, we employed the `tc` command in conjunction with a local network setup.

In this evaluation, we focus on the maliciously secure version. We consider two values of ℓ_1 in Table 1 with $q_{\text{on}} = 2^{96}$; $\ell_1 = 109$ for $n' = 5 \cdot m$ for $m \approx 1.3n$, and $\ell_1 = 150$ for $n' = 2 \cdot m$ for $m \approx 1.3n$.

Main Comparison Target. Our main comparison target is the state-of-the-art OPRF protocol [10] in the RS construction (fast version) that using LPN-based VOLE over $\text{GF}(2^{128})$. The performance of those performance strongly depends on the specific choice of the code for LPN. We consider quasi-cyclic LDPC code [5] which we denote below by QC, and a new family of codes named expand-convolute codes proposed by Rindal et al. [11], especially ExConv7x24 (resp. ExConv21x24) implemented in libOTe by EC1 (resp. EC2).

Remark 3. There was another recent proposal of code family called Silver [6] that retains quite aggressive structure to have blazing-fast performance. However, we do not compare with this family, as it turns out to be vulnerable [11].

Concrete Linear Codes. Although Table 2 provides the minimal length of n_c such that $[n_c, \ell_1, \lceil 128/f \rceil]_2$, the concrete construction of such linear code is

Table 3. Specification of the binary linear codes for our experiments.

ℓ_1	f	n_c	Binary code
109	1	550	RS[50, 19, 32] ₆₄ + [11, 6, 4] ₂
	2	350	RS[50, 19, 32] ₆₄ + [7, 6, 2] ₂
	4	238	RS[34, 19, 16] ₆₄ + [7, 6, 2] ₂
	6	192	RS[32, 22, 11] ₃₂ + [6, 5, 2] ₂
	8	174	RS[29, 22, 8] ₃₂ + [6, 5, 2] ₂
150	1	616	RS[56, 25, 32] ₆₄ + [11, 6, 4] ₂
	2	392	RS[56, 25, 32] ₆₄ + [7, 6, 2] ₂
	4	280	RS[40, 25, 16] ₆₄ + [7, 6, 2] ₂
	6	245	RS[35, 25, 11] ₆₄ + [7, 6, 2] ₂
	8	224	RS[32, 25, 8] ₆₄ + [7, 6, 2] ₂

not known for most cases. Thus, for our experiments, we use the binary linear code constructed by combining a Reed-Solomon (RS) code and another binary code, which method is already used in [9]: To be precise, given ℓ_1 -bit input, we first apply some RS code $\text{RS}[n_{\text{rs}}, k_{\text{rs}}, d_{\text{rs}}]_q$ by embedding ℓ_1 -bit inputs into k_{rs} elements of $\text{GF}(q)$, which outputs n_{rs} elements of $\text{GF}(q)$. After then, we apply $[n_b, \log q, d_b]_2$ code for each $\text{GF}(q)$ element by understanding it as $\log q$ -bits. This implies a binary code $[n_{\text{rs}} \cdot n_b, \ell_1, d_{\text{rs}} \cdot d_b]_2$. Given ℓ_1 and d_c , we exhaustively searched through all possible RS codes and binary linear codes to find the one satisfying $d_{\text{rs}} \cdot d_b = d_c$ with the shortest $n_{\text{rs}} \cdot n_b$, and the concrete codes are obtained from SageMath. The results are summarized in Table 3.

Remark 4. There would be another construction of linear code $[n_c, \ell_1, d_c]$ shorter than our found RS code & binary code combination; for example, it is known a construction of $[164, 109, 16]_2$ linear code [1] for $\ell_1 = 109$ and $f = 8$ case, which is 10-bit shorter than our construction $[174, 110, 16]_2$. One might try to replace our codes by investigating further shorter linear code, which directly improves the performance without any harm on security.

Communication Costs. Recall that the transmission of the correction matrix $U' \in \mathbb{B}^{m \times n_c}$ dominates the communication cost, precisely $n_c \cdot b \cdot (1 + \epsilon_{\text{okvs}})n_x$ bits. Thus the communication cost would be totally proportional to $n_c \cdot b$, which is n_c in our case, and $b = 128$ in [10]. Table 4 shows the communication cost based on the concrete linear codes found in Table 3, and one can check that indeed the communication cost is exactly proportional to $n_c \cdot b$. We have to say that [10] requires smaller communication than ours, as it has $n_c \cdot b = 128$. However, we would like to remark that the $f = 1$ case is actually corresponds to the original PRTY construction [9] while replacing the subroutines to latest one. In this view, our generalized f choice narrows the gap between [10] protocols

and our one: for example $\ell_1 = 109$ (or $n' = 5m$), the communication cost gap between Minicrypt and LPN drops to 4.2x ($f = 1$) to 1.3x ($f = 8$).

Table 4. Total communication costs of our OPRF and [10] for $n = 2^{20}$ items. The ‘ n'/m ’ row shows the number of PRF values that malicious receiver can obtain. Our protocols can set $q_{\text{on}} = 2^{96}$ and $q_{\text{off}} = 2^{128}$ thanks to our mitigation, which is not applicable to the previous work.

$n = 2^{20}, m \approx 1.3n$		Ours		[10]
n'/m		5	2	5.6
ℓ_1		109	150	128
Communication (MB)	$f = 1$	93.56	104.8	
	$f = 2$	59.56	66.71	22.23(QC)
	$f = 4$	40.53	47.68	22.78(EC1)
	$f = 6$	32.71	41.75	22.42(EC2)
	$f = 8$	29.67	38.20	

Running Time. To check the total running time of OPRF in various network setting, we used 3 settings. The first one is 100 Mbps with 100 ms rtt (round trip time), and the second one is 1 Gbps with 1 ms rtt, the last one is 5 Gbps with 1 ms rtt. These settings are done by using linux `tc` command in local host network. We further note that the public implementation [16] of [10] only supports fixed $\ell_1 = 128$ now, which translates to $n' = 5.6m$. As a fair comparison, we provide the performances for $\ell_1 = 109$ case of $n' = 5m$ in Table 5.

Table 5. Total running time (in second) of our OPRF and [10] on various network settings for $n = 2^{20}$ items. Under the parameters used in this evaluation, our protocol allows at most $n' = 5m$ PRF evaluations, and [10] allows at most $n' = 5.6m$ PRF evaluations.

$n = 2^{20}$		100 Mbps	1 Gbps	5 Gbps	Assumption
Ours	$f = 1$	14.2	2.62	2.03	Minicrypt
	$f = 2$	10.0	2.18	1.80	
	$f = 4$	7.787	2.25	1.99	
	$f = 6$	7.783	2.96	2.74	
	$f = 8$	9.37	4.75	4.55	
[10]	QC	5.304	2.32	2.31	dual-LPN
	EC1	4.647	0.990	0.980	
	EC2	5.056	1.629	1.577	

Table 5 shows OPRF running time with $f = 1, 2, 4, 6, 8$ at our construction and the previous work [10]. The implementation of the previous work is in public [16], so we re-run the code in our evaluation setting for fair comparison. In the fast (5 Gbps and 1 Gbps) network environments, our $f = 2$ case is even faster than [10] with QC. Since our protocol only requires Minicrypt assumption while QC is not, we might say our protocol is strictly better than it.

In slower network (100 Mbps) environment where the communication cost affects a lot, the protocols of [10] are clearly still better. However, we can confirm again the generalized field choice helps to narrow the gap between [10] protocols and our one. Specifically, the $f = 1$ case that corresponds to revised PRTY construction takes 14.2s, which is $2.7 - 3x$ than [10] protocols, while our $f = 6$ of 7.783s decreases the gap by $1.5 - 1.7x$.

Discussion with Breakdown. Table 6 shows the breakdown for of our protocol timing results in Table 5. The ‘Transpose V ’ row is an implementation-specific part, which cannot be seen in Fig. 9: The VOLE outputs $V, W \in \mathbb{F}^{m \times n_c}$ are obtained by n_c independent call of VOLE, and they are naturally arranged in a column-wise manner. However, the OKVS decoding understands V (and W) as a length m vector of $n_c \cdot \log |\mathbb{F}|$ -bits, we need to transpose them. As Table 6 shows, such transpose also takes quite a large computational cost.

Table 6. Breakdown of our protocol timings (in second) in Table 5.

f	100 Mbps			1 Gbps			5 Gbps		
	1	4	8	1	4	8	1	4	8
VOLE	0.670	0.822	2.673	0.246	0.430	2.333	0.244	0.414	2.324
Transpose	0.133	0.263	0.664	0.132	0.265	0.675	0.134	0.263	0.667
OKVS Encode	0.356	0.350	0.348	0.356	0.361	0.354	0.352	0.354	0.354
Apply Lin. Code	0.373	0.114	0.100	0.378	0.116	0.100	0.379	0.116	0.101
Send/Recv Corr.	12.145	5.568	4.580	0.995	0.418	0.331	0.404	0.182	0.140
OKVS Decode	0.343	0.491	0.615	0.332	0.485	0.620	0.336	0.485	0.627

We finally remark that the first two steps (VOLE and transpose) can be done in offline, before the input set X is determined. In other words, they can be understood as a setup phase before the main OPRF starts. Concretely, for $f = 8$, the first two steps occupy from 77% in the total timings un the 5 Gbps network. Thus, we can conclude that our protocol with $f = 8$ would be a nice choice for the situations where some offline computation is allowed, probably even better than [10] protocol with EC-LPN-based VOLE.

Acknowledgments. The authors would like to thank Peter Rindal for helpful comments and discussions, especially for verifying the early discovery of the flaw and providing clear understanding on the current state of OPRF and PSI protocols.

A Details for RB-OKVS Overfitting Attack

Case $n' < 2m$. Similarly to PaXoS, we reduce the $(n, \frac{km}{k-1})$ -overfitting game to a k -XOR problem for $k > 2$ as follows. As solving a k -XOR problem of $(d + \ell_1)$ -bit strings costs $O(k2^{\frac{d+\ell_1}{1+\lceil \log k \rceil}})$ time, our attack costs $O(m2^{\frac{d+\ell_1}{1+\lceil \log k \rceil}})$ time including time for querying to the random oracle.

1. Let $Q = \{x_1, \dots, x_q\}$ be an arbitrary subset in $\{0, 1\}^*$. Query all the items in Q to h_0, h_1 , and H_1 . Bucketize Q by h_1 , denoting B_{h_1} the corresponding bucket. Then, there are m buckets, and there are expectedly q/m items per bucket. For simplicity, we assume that $(k-1)|d$ and $(k-1)|m$.
2. For $j \in \{1, k, \dots, m-d-k+2\}$, do the following with initialization $X' \leftarrow \{\}$.
 - (a) Solve a k -XOR problem for $h_0(x)||H_1(x)$ in buckets B_j . If leftmost $k \times (k-1)$ -submatrix has rank strictly less than $k-1$, then repeat this step to find another solution.
 - (b) Gather the proper solutions of the k -XOR problem to X' .
3. For $j \in \{m-d+1, m-d+k, \dots, m-k+2\}$, do the following with initialization $Y \leftarrow \{\}$.
 - (a) Solve a k -XOR problem for $h_0(x)||H_1(x)$ in buckets B_{m-d+1} and let x'_1, \dots, x'_k are solutions.
 - (b) Make a matrix $K \in \mathbb{F}_2^{k \times (m+\ell_1)}$ from $\text{row}(x_i)||H_1(x_i)$.
 - (c) Let K' be a submatrix of K in $\mathbb{F}_2^{k \times (k-1)}$, whose i -th column is equal to the $(j+i-1)$ -th column of K .
 - (d) If $\text{rank}(K') = k-1$, then add x'_1, \dots, x'_k to Y . Otherwise, repeat step 3 for same j .
4. Denote $X' = \{x'_1, \dots, x'_{n'}\}$ where $n' = k \cdot \frac{m-d}{k-1} + k \cdot \frac{d}{k-1} = \frac{km}{k-1}$. Since $\text{rank}(\text{row}(\vec{X}')) = \text{rank}(\text{row}(\vec{X}')|H_1(\vec{X}'))$, there is a solution P' of the linear equation $\text{row}(X') \cdot P' = H_1(X')$.

Case $n' \geq 2m$. Similarly to PaXoS, we reduce the OKVS overfitting problem to a c -multicollision finding problem as follows. As finding c -multicollision of $(d + \ell_1)$ -bit strings costs $O((c!)^{1/c} 2^{\frac{(c-1)(d+\ell_1)}{c}})$ time, our attack costs $O(m2^{\frac{(c-1)(d+\ell_1)}{c}})$ time for small enough c .

1. Let $Q = \{x_1, \dots, x_q\}$ be an arbitrary subset in $\{0, 1\}^*$. Query all the items in Q to h_0, h_1 , and H_1 . Bucketize Q by h_1 , denoting B_{h_1} the corresponding bucket. Then, there are m buckets, and there are expectedly q/m items per bucket.
2. Let $k > 2$. For $j \in \{1, 2, \dots, m-d\}$, do the following with initialization $X' \leftarrow \{\}$.
 - (a) Find a c -multicollision for $h_0(x)||H_1(x)$ from bucket B_j . If the first bit of $h_0(x)$ is 0, find another solution.
 - (b) Gather the proper solutions of the problem to X' .

3. For $j \in \{m - d + 1, m - d, \dots, m\}$, do the following with initialization $Y \leftarrow \{\}$.
 - (a) Find a c -multicollision for $h_0(x) \| H_1(x)$ in buckets B_{m-d+1} and let x'_1, \dots, x'_c are those solutions.
 - (b) Add proper solutions x'_1, \dots, x'_c to Y .
4. If $\text{row}(Y)$ has rank less than d , go back to Step 3. Otherwise, $X' \leftarrow X' \cup Y$.
5. Denote $X' = \{x'_1, \dots, x'_{n'}\}$ where $n' = cm$. Since $\text{rank}(\text{row}(\vec{X}')) = \text{rank}(\text{row}(\vec{X}') | H_1(\vec{X}'))$, there is a solution P' of the linear equation $\text{row}(X') \cdot P' = H_1(X')$.

References

1. Code Tables: Bounds on the parameters of various types of codes (2022), <http://codetables.de/>
2. Baum, C., Braun, L., de Saint Guilhem, C.D., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures from VOLE-in-the-Head. In: Handschuh, H., Lysyanskaya, A. (eds.) *Advances in Cryptology – CRYPTO 2023*. pp. 581–615. Springer Nature Switzerland, Cham (2023)
3. Bienstock, A., Patel, S., Seo, J.Y., Yeo, K.: Near-Optimal Oblivious Key-Value Stores for Efficient PSI, PSU and Volume-Hiding Multi-Maps. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 301–318. USENIX Association, Anaheim, CA (Aug 2023), <https://www.usenix.org/conference/usenixsecurity23/presentation/bienstock>
4. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Resch, N., Scholl, P.: Correlated Pseudorandomness from Expand-Accumulate Codes. In: Dodis, Y., Shrimpton, T. (eds.) *Advances in Cryptology – CRYPTO 2022*. Springer Nature Switzerland, Cham (2022)
5. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. In: *CCS 2019*. pp. 291–308 (2019)
6. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes. In: *CRYPTO 2021*. pp. 502–534. Springer, Cham (2021)
7. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: *CRYPTO 2021*. pp. 395–425. Springer (2021)
8. Orrù, M., Orsini, E., Scholl, P.: Actively Secure 1-out-of-N OT Extension with Application to Private Set Intersection. In: Handschuh, H. (ed.) *Topics in Cryptology – CT-RSA 2017*. pp. 381–396. Springer International Publishing, Cham (2017)
9. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from PaXoS: fast, malicious Private Set Intersection. In: *EUROCRYPT 2020*. pp. 739–767. Springer (2020)
10. Raghuraman, S., Rindal, P.: Blazing Fast PSI from Improved OKVS and Subfield VOLE. In: *CCS 2022*. pp. 2505–2517. ACM, New York, NY, USA (2022)
11. Raghuraman, S., Rindal, P., Tanguy, T.: Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN. In: Handschuh, H., Lysyanskaya, A. (eds.) *Advances in Cryptology – CRYPTO 2023*. Springer Nature Switzerland, Cham (2023)
12. Rindal, P.: libOTe: an efficient, portable, and easy to use Oblivious Transfer Library (2022), <https://github.com/osu-crypto/libOTe>

13. Rindal, P., Schoppmann, P.: VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE. In: EUROCRYPT 2021. pp. 901–930. Springer, Cham (2021)
14. Roy, L.: SoftSpokenOT: Quieter OT Extension from Small-Field Silent VOLE in the Minicrypt Model. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13507, pp. 657–687. Springer (2022). https://doi.org/10.1007/978-3-031-15802-5_23, https://doi.org/10.1007/978-3-031-15802-5_23
15. Suzuki, K., Tonien, D., Kurosawa, K., Toyota, K.: Birthday Paradox for Multi-collisions. In: Rhee, M.S., Lee, B. (eds.) Information Security and Cryptology – ICISC 2006. pp. 29–40. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
16. Visa-Research: volepsi: Efficient private set intersection base on vole (2022), <https://github.com/Visa-Research/volepsi>
17. Wagner, D.: A Generalized Birthday Problem. In: Yung, M. (ed.) Advances in Cryptology — CRYPTO 2002. pp. 288–304. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

Quantum Cryptanalysis



Reducing the Number of Qubits in Quantum Information Set Decoding

Clémence Chevnard^(✉), Pierre-Alain Fouque, and André Schrottenloher

Univ Rennes, Inria, CNRS, IRISA, Rennes, France

{clemence.chevnard,pierre-alain.fouque,andre.schrottenloher}@inria.fr

Abstract. This paper presents an optimization of the memory cost of the quantum *Information Set Decoding* (ISD) algorithm proposed by Bernstein (PQCrypto 2010), obtained by combining Prange’s ISD with Grover’s quantum search.

When the code has constant rate and length n , this algorithm essentially performs a quantum search which, at each iteration, solves a linear system of dimension $\mathcal{O}(n)$. The typical code lengths used in post-quantum public-key cryptosystems range from 10^3 to 10^5 . Gaussian elimination, which was used in previous works, needs $\mathcal{O}(n^2)$ space to represent the matrix, resulting in millions or billions of (logical) qubits for these schemes.

In this paper, we propose instead to use the algorithm for sparse matrix inversion of Wiedemann (IEEE Trans. inf. theory 1986). The interest of Wiedemann’s method is that one relies only on the implementation of a matrix-vector product, where the matrix can be represented in an implicit way. This is the case here.

We give two main trade-offs, which we have fully implemented, tested on small instances, and benchmarked for larger instances. The first one is a quantum circuit using $\mathcal{O}(n)$ qubits, $\mathcal{O}(n^3)$ Toffoli gates like Gaussian elimination, and depth $\mathcal{O}(n^2 \log n)$. The second one is a quantum circuit using $\mathcal{O}(n \log^2 n)$ qubits, $\mathcal{O}(n^3)$ gates in total but only $\mathcal{O}(n^2 \log^2 n)$ Toffoli gates, which relies on a different representation of the search space.

As an example, for the smallest Classic McEliece parameters we estimate that the Quantum Prange’s algorithm can run with 18098 qubits, while previous works would have required at least half a million qubits.

Keywords: Prange’s Algorithm · Quantum Search · Information Set Decoding · Quantum Cryptanalysis

1 Introduction

Since the start of the NIST’s post-quantum cryptography standardization process [42], several cryptosystems based on error correcting codes came to light. The remaining key-encapsulation candidates Classic McEliece [12], HQC [1] and BIKE [3] are currently in the fourth round [44]. Additionally, many more code-based cryptosystems have been proposed at the NIST’s call for additional post-quantum signature schemes [43].

Cryptographic algorithms based on codes usually rely on the hardness of the Syndrome Decoding Problem (SDP). This problem essentially consists in finding a solution to an undetermined linear system, which is constrained under a given metric. In the cases considered on this paper, the Hamming metric is used, which counts the number of non-zero coordinates.

The most efficient algorithms to solve the SDP are the family of Information Set Decoding algorithms (ISD), starting with Prange’s algorithm [47], which have been gradually improved over time with list-merging subroutines [6, 37, 39, 52], nearest-neighbor techniques [15, 40] and more recently sieving techniques [25, 32]. However, all these optimizations essentially improve the time complexity at the detriment of the memory complexity.

Principle. Let \mathbf{H} be the parity-check matrix of the code, which has n columns and $n - k$ rows, where n is the length of the code and k its dimension (the codewords forming its kernel). The goal of SDP is to find a vector \mathbf{s} such that $\mathbf{H}\mathbf{s}$ has some prescribed Hamming weight.

Prange’s algorithm selects at random a subset I of $n - k$ columns of \mathbf{H} , which defines a square submatrix \mathbf{H}_I , and inverts the subsystem defined by \mathbf{H}_I . If the nonzero coefficients of the vector solution to SDP correspond to columns of \mathbf{H} that are all in \mathbf{H}_I , then inverting the subsystem defined by \mathbf{H}_I will allow to retrieve this solution. And since \mathbf{H}_I is square, Gaussian elimination can be used to invert the system, which will have few solutions on average. This procedure is repeated for random choices of columns I until a solution is found.

Quantum ISD. In the quantum setting, one can speedup this algorithm using Grover’s quantum search, as proposed by Bernstein [10] (an algorithm that we will call “quantum Prange” in what follows). Indeed, the subsets of columns I form a well-defined search space, and solving the subsystem \mathbf{H}_I allows to test if I is solution to our problem. Using Grover’s search, the number of iterations decreases to a square root of its classical value. Despite this asymptotic speedup, we should note that in practice, the cost of solving \mathbf{H}_I , though a polynomial factor, is far from negligible.

Similarly to the classical setting, improved quantum ISD algorithms were introduced later on by Kachigar and Tillich [34] and Kirshanova [36] using various techniques from classical ISD, notably quantum versions of the MMT [39] and BJMM [6] algorithms using quantum walks. More recently, Kimura et al. [35] presented another trade-off between time and memory using a combination of Kirshanova’s algorithm, Both and May’s classical ISD algorithm [15], and Grover’s algorithm. Chailloux et al. [19] extended the scope of quantum Prange by adapting it to other metrics than the Hamming one. Since we are particularly targeting the memory complexity of quantum ISD, and since we consider the Hamming case only, the improvements of [19, 34–36] will not be considered further in this paper.

Improving Quantum Prange. Esser et al. [26, 27] introduced several improvements to quantum Prange, leading to polynomial runtime improvements: first,

an optimization using the Lee-Brickell algorithm [37] and an optimization based on preprocessing the parity-check matrix to systematic form, which we will not reuse here.

In [27], motivated by the large number of qubits required, they proposed a hybrid quantum-classical trade-off. The idea is to guess a first part of the 0 coefficients' positions in a precomputation on a classical computer, and then to carry a smaller instance of the quantum Prange's algorithm. Another possible optimization, in the same fashion, is to not consider all of the equations in the square systems one aims at solving. The combination of these two optimizations reduces the qubit cost by a constant, but increases the time exponentially.

More recently, Perriello et al. [46] performed complete quantum cost estimates of quantum Prange, which we will compare to for the parameters of the NIST code-based candidates.

Memory Complexity and Comparisons. The parity-check matrix \mathbf{H} itself, and its sub-matrices that one tries to invert, occupies a space $\mathcal{O}(n^2)$. Therefore the memory complexity of Prange's algorithm, classical or quantum, is not negligible. In code-based cryptosystems, n ranges between 10^3 and 10^5 , leading to millions or billions of logical qubits. This is much more than the thousands required by Shor's algorithm for factoring RSA semiprimes [30] or even exhaustive search of AES keys [33], which is explicitly used by the NIST as a benchmark for post-quantum security levels [42].

Looking back at the optimizations of quantum ISD [26, 27, 46], the number of qubits could never decrease below $\mathcal{O}(n^2)$, despite improvements in the constants. This is simply because these quantum algorithms used Gaussian elimination to invert the sub-matrices considered during the Grover search iterations.

At the start of the post-quantum standardization process [42], the NIST focused on the metrics of gate count and depth. Specifically, they suggested to compare total gate counts under a limitation of circuit depth (i.e., total running time), denoted MAXDEPTH, which ranges from 2^{40} to 2^{96} . Under these metrics, the total number of qubits is not a limiting parameter, since one expects the algorithms to become massively parallel. However, the number of qubits that each individual machine uses remains of significance. Reducing this amount would allow more flexible trade-offs for such parallel searches.

Contribution and Organization. In this paper, we optimize the qubit count of quantum ISD. Our innovation lies in the way the sub-matrices considered throughout the quantum search are represented, and inverted.

We use Wiedemann's matrix inversion algorithm [53]. This technique, in our case, is interesting because it reduces the inversion of a dimension- n matrix to computing a series of $\mathcal{O}(n)$ matrix-vector products, and using the Berlekamp-Massey algorithm on linear recurrent sequences of size $\mathcal{O}(n)$. We show how to implement the matrix-vector product by a sub-matrix \mathbf{H}_I given a compact representation of the choice I of columns. Next, we give a quantum implementation of the Berlekamp-Massey algorithm over \mathbb{F}_2 using $\mathcal{O}(n)$ space only. This gives us a circuit for the Grover iteration in quantum Prange that uses $\mathcal{O}(n)$ qubits.

This first implementation is optimized for space, but the resulting gate count $\mathcal{O}(n^3)$ and circuit depth $\mathcal{O}(n^2 \log n)$ suffer from large constant factors. This motivates us to give another trade-off, by changing the representation of I in the algorithm. In the first version, I is represented as a *selection* of columns of the matrix \mathbf{H} , i.e., a vector of length n and weight $n - k$, similarly to what is commonly done in previous works. In the second version, I is represented as a permutation of the columns (an idea that appeared in [45], but without further details). This permutation is implemented using a *switching network* with $\mathcal{O}(n \log^2 n)$ switches. The qubit count increases to $\mathcal{O}(n \log^2 n)$, but this is compensated by a significant improvement in gate count.

First of all, we reduce the depth to $\mathcal{O}(n^2)$, becoming asymptotically optimal, and the Toffoli gate count to $\mathcal{O}(n^2 \log^2 n)$ (while the total gate count remains $\mathcal{O}(n^3)$), improving over circuits based on Gaussian elimination. This is particularly interesting from an implementation standpoint, as Toffoli gates, which contain nonlinearity, are considered much harder to implement than NOT and CNOT gates. This also shows that there are further advantages to Wiedemann inversion than just space. Finally, when the parity-check matrix is block-circulant, as in BIKE and HQC, we show that the *total* gate count of the Grover iteration can be reduced to $\mathcal{O}(n^2 \log^2 n)$ (Theorem 4), though our current implementation achieves only a minor gain, at the expense of circuit depth.

Inverting large sparse or implicit matrices is a building block of other quantum algorithms, notably quantum algorithms for solving multivariate quadratic systems [13, 28]. The space complexity was not discussed in [28], and in [13], the generic Bennett-Tompa reversibility trade-off is used [7], which leads to additional non-negligible factors in time and space. Our results could lead to improvements in both cases, although this would require a dedicated analysis.

On a side note, it has been pointed out by a reviewer that our space optimization of quantum ISD is possible because the quantum circuit is *instance-dependent*, i.e., the parity-check matrix \mathbf{H} is encoded in the circuit instead of being represented as an input. This is not an uncommon occurrence in quantum cryptanalysis: typical implementations of Shor’s algorithm [30] use a precomputation which also makes the circuit instance-dependent. However, it might be interesting to develop further such techniques to reduce the qubit count.

Organization of the Paper. Section 2 describes Prange’s algorithm and Wiedemann’s inversion [53]. In Sect. 3 we detail the quantum version of Prange’s algorithm, and explain how to replace the linear algebra part with Wiedemann’s algorithm. Our main result is stated here, but its proof spans the following sections.

First, in Sect. 4, we give our implementation of Wiedemann inversion, including a quantum reversible circuit for the Berlekamp-Massey algorithm [8]. This implementation uses as black-box a quantum algorithm for matrix-vector product. Then, in Sect. 5, we propose two such implementations. The first one (space-optimized) reduces the space to $\mathcal{O}(n)$. The second (Toffoli-optimized) offers a trade-off with a space $\mathcal{O}(n \log^2 n)$ and a reduced Toffoli cost. We give precise

formulas for the costs of all these circuits. Finally, in Sect. 6, we evaluate these costs for the parameters of code-based cryptosystems.

We implemented the quantum circuits considered in this paper using the Qiskit framework [48]. Our code is available at <https://gitlab.inria.fr/capsule/quantum-isd-less-qubits>.

2 Preliminaries

Throughout this paper, n and $k \leq n$ are integers. We use bold notations for vectors ($\mathbf{e}, \mathbf{s}, \dots$) and uppercase letters for matrices ($\mathbf{A}, \mathbf{H}, \dots$). The transpose of a matrix is denoted \mathbf{A}^T . Since we work only in \mathbb{F}_2 , addition $+$ will always refer to addition in \mathbb{F}_2 (binary XOR) or in a vector space over \mathbb{F}_2 . The symbol δ_{ij} corresponds to the Kronecker delta function, that equals 1 if and only if $i = j$, and 0 otherwise.

For q a prime, a *linear code* of length n and dimension k over \mathbb{F}_q is a k -dimensional vector subspace \mathbb{F}_q^n , which can be defined as the kernel of a *parity-check* matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$. In this paper we consider the case of *binary* codes, i.e., $q = 2$. The Hamming weight hw is defined, for vectors of any length, by the number of non-zero coordinates. We consider the Syndrome Decoding problem $SD(n, k, w)$: given as input $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ and $\mathbf{s} \in \mathbb{F}_2^{n-k}$ (the error syndrome), find $\mathbf{e} \in \mathbb{F}_2^n$ such that $\text{hw}(\mathbf{e}) = w$.

While the decision version of this problem, i.e., the existence of such a solution, is well-known to be NP-complete [9], we consider the search version where \mathbf{H} is sampled uniformly at random, and the existence of the solution is guaranteed. With proper choice of the parameters n, k, w , this case is still believed to be hard for classical and quantum computers alike.

Problem 1 (Random Decoding). Given as input $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ sampled uniformly at random, and $\mathbf{s} = \mathbf{H}\mathbf{e}$ where \mathbf{e} is sampled uniformly at random from vectors of weight w , find \mathbf{e} .

2.1 Prange's Algorithm

In [47], Prange introduced an algorithm for generic decoding (Algorithm 1) which forms the basis of the family of Information Set Decoding (ISD) algorithms. We use it to solve Problem 1. The idea of the algorithm is to select a random subset of the columns of \mathbf{H} . Let $S_{n,k}$ be the set of all subsets of $\{0, \dots, n-1\}$ with $n-k$ elements.

Given $I \in S_{n,k}$, we define \mathbf{H}_I as the sub-matrix of \mathbf{H} which keeps these columns only. If all non-zero positions of the vector \mathbf{e} are in I , the equation system $\mathbf{H}_I \mathbf{x} = \mathbf{s}$ admits a solution \mathbf{x} of weight w , and extending this solution by zeroes provides a solution to the SDP¹.

¹ Note that we have considered here the binary case only; the case of a generic q requires more care.

Algorithm 1. Prange’s algorithm, binary case [47].

Input: parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$, syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$, weight $w \leq n - k$
Output: vector \mathbf{e} such that $\mathbf{H}\mathbf{e} = \mathbf{s}$ and $\text{hw}(\mathbf{e}) = w$
repeat
 Choose a random subset $I \in S_{n,k}$
 If \mathbf{H}_I is not invertible, **continue**
 Solve the linear system $\mathbf{H}_I \mathbf{e}_I = \mathbf{s}$ for \mathbf{e}_I
until $\text{hw}(\mathbf{e}_I) = w$
Return \mathbf{e} = extension of \mathbf{e}_I with zeroes

Runtime. As a first remark, it is well known that a random square matrix in \mathbb{F}_2 is invertible with probability at least 0.288 [21]. In order to bound easily the runtime of Prange’s algorithm, we make the following heuristic assumption.

Heuristic 1. A matrix \mathbf{H}_I , where I leads to the solution, is invertible with probability 0.288.

Intuitively, the invertibility of the sub-matrix \mathbf{H}_I should be independent from whether I is a solution or not. While we cannot exclude pathological values of \mathbf{H} and \mathbf{s} , we can assume that such cases occur with negligible probability.

Theorem 1. *Under Heuristic 1, Algorithm 1 succeeds with $\frac{1}{p}$ iterations on average, where:*

$$p = 0.288 \frac{\binom{n-k}{w}}{\binom{n}{w}}, \tag{1}$$

and has a time complexity $\mathcal{O}(\frac{1}{p}(n - k)^\omega)$ where ω is the matrix multiplication exponent.

Proof. A first important remark is that the algorithm does succeed: this is because there are in general many solutions I , and by Heuristic 1, one of them will lead to an invertible sub-matrix – so it will eventually be found.

Consider the columns H_{i_1}, \dots, H_{i_w} of \mathbf{H} that correspond to the non-zero coefficients of the solution \mathbf{e} . The loop succeeds whenever $\{i_1, \dots, i_w\} \subseteq I$ and \mathbf{H}_I is invertible. By Heuristic 1, the number of good choices for I is: $0.288 \binom{n-w}{n-k-w}$, while the total number of possible I is $\binom{n}{n-k}$. This gives a probability of success in the loop $p = 0.288 \frac{\binom{n-w}{n-k-w}}{\binom{n}{n-k}} = 0.288 \frac{\binom{n-k}{w}}{\binom{n}{w}}$.

The average number of iterations before achieving a success follows from this. Each iteration requires to invert a linear system of dimension $n - k$, which requires $\mathcal{O}((n - k)^\omega)$ elementary operations. \square

The subsequent improvements to Prange’s ISD [6, 15, 25, 32, 37, 39, 40, 52] put less constraints on I , which decreases the number of loop iterations; however they increase the complexity of recovering \mathbf{e} during the iteration. Importantly, any improvement in the time complexity exponent comes at the expense of a larger memory complexity, which is why we do not consider these variants in what follows.

2.2 Wiedemann's Inversion Algorithm

This section follows Wiedemann [53].

Finding Relations. The Berlekamp-Massey algorithm [8,38] takes as input a sequence $(s_0, \dots, s_{N-1}) \in \mathbb{F}_q^N$ satisfying a linear recurrence relation over \mathbb{F}_q :

$$\exists \ell \leq N, \exists c_0 \neq 0, c_1, \dots, c_\ell, \forall i \leq N, c_0 s_i + c_1 s_{i-1} + \dots + c_\ell s_{i-\ell} = 0 . \quad (2)$$

This relation is represented by a polynomial $C(X) := c_0 + c_1 X + \dots + c_\ell X^\ell \in \mathbb{F}_q[X]$, of degree ℓ . Assuming that $2\ell \leq N$, the Berlekamp-Massey algorithm returns such a polynomial $C(X)$ with the smallest possible degree. We defer the details of this algorithm to Sect. 4.2. It runs in $\mathcal{O}(N^2)$ operations, using $\mathcal{O}(N)$ space. Using its similarity to an extended Euclidean algorithm on polynomials [24], this time can be reduced to $\tilde{\mathcal{O}}(N)$, but this will not be important for us as the time complexity of Berlekamp-Massey will not be dominant in our algorithms.

Wiedemann's Algorithm. Wiedemann [53] considers the problem to invert a sparse matrix over \mathbb{F}_q , or more generally, any matrix \mathbf{A} for which only a black-box matrix-vector product is provided. In the case of sparse matrices, this is motivated by the efficiency of such products, and the low space complexity.

Given inputs \mathbf{A} and $\mathbf{s} \in \mathbb{F}_q^n$, the goal is to find the unique \mathbf{x} such that $\mathbf{A}\mathbf{x} = \mathbf{s}$. In the following, we assume that \mathbf{A} is invertible. Wiedemann [53] provides further analysis to deal with non-invertible matrices, but this will not be necessary for us, as we will essentially need to succeed with constant probability for random matrices (therefore, succeeding for invertible matrices is sufficient for us).

Let S be the space spanned by $\{\mathbf{A}^i \mathbf{s}, i \in \mathbb{N}\}$ where $\mathbf{A}^0 = \mathbf{I}$ is the identity matrix. We consider the action of \mathbf{A} on this space, defined by an operator A_S with minimal polynomial $P(X) \in \mathbb{F}_q[X]$. The polynomial P is normalized to have its first coefficient equal to 1. Let $Q(X) = (1 - P(X))/X \in \mathbb{F}_q[X]$, which is of degree $n - 1$ at most. We have:

$$P(\mathbf{A})\mathbf{s} = \mathbf{0} \implies \mathbf{A}(Q(\mathbf{A})\mathbf{s}) = \mathbf{s} \implies \mathbf{x} = Q(\mathbf{A})\mathbf{s} . \quad (3)$$

Given Q , evaluating $Q(\mathbf{A})\mathbf{s}$ can be done by a series of n matrix-vector products and $\mathcal{O}(n)$ temporary space, by Horner's method. Therefore, the search for \mathbf{x} is reduced to the search for P .

The search for P can be reduced to finding linear recurrences, as follows. Since evaluating $\mathbf{A}^i \mathbf{s}$ yields a sequence of vectors in \mathbb{F}_q^n , and not of scalars, one selects a (random) vector $\mathbf{u} \in \mathbb{F}_q^n$ and computes the sequence of projections: $(\mathbf{u}^T \mathbf{A}^i \mathbf{s}, i \in \mathbb{N})$. This sequence satisfies a linear recurrence, with a minimal polynomial $C(X)$ that divides $P(X)$. Since P is of degree n , only $2n$ terms need to be computed. In fact, after on expectation $\mathcal{O}(\log n)$ tries with random vectors \mathbf{u} , one will obtain $C(X)$. But it is possible to arrive at this result faster using a slightly more technical algorithm which finds first a divisor C_0 of P , then reduces the problem to finding P/C_0 , etc. This is summarized in Algorithm 2.

The probability of success of this algorithm follows Lemma 1.

Algorithm 2. Wiedemann’s algorithm for inversion.

Input: invertible matrix \mathbf{A} accessed only by a black-box product operator: $\mathbf{y} \mapsto \mathbf{A}\mathbf{y}$; vector \mathbf{s}
Output: $\mathbf{x} = \mathbf{A}^{-1}\mathbf{s}$

- 1: $\mathbf{t} \leftarrow \mathbf{s}, \mathbf{y} \leftarrow 0$
- 2: $d \leftarrow 0$ ▷ Current degree of the polynomial
- 3: **repeat**
- 4: Select \mathbf{u} uniformly at random
- 5: Compute the first $2(n - d)$ terms of the sequence $\mathbf{u}^T \mathbf{A}^i \mathbf{t}$
- 6: Compute $C(X)$, the minimal polynomial of this sequence
- 7: Let $C'(X) = (C(X) - 1)/X$
- 8: $\mathbf{y} \leftarrow \mathbf{y} + C'(\mathbf{A})\mathbf{t}$
- 9: $\mathbf{t} = \mathbf{s} + \mathbf{A}\mathbf{y}$
- 10: $d \leftarrow d + \deg(C)$
- 11: **until** $\mathbf{t} = 0$
- 12: **Return** $-\mathbf{y}$

Lemma 1 (From [53], Section VI). *For $k > 1$, the probability that after k iterations in the main loop of Algorithm 2, one has $\mathbf{t} = 0$ (and $\mathbf{A}(-\mathbf{y}) = \mathbf{s}$), is lower bounded by:*

As a constant success probability will be enough for us, we can run Algorithm 2 with a constant number of loops. In particular with $q = 2$, by using $k = 2$ we ensure a probability of success bigger than $2^{-1.70}$. As proposed by Wiedemann, we will also replace the selected vectors \mathbf{u} by deterministic unit vectors, which consist merely in selecting the first and second coordinates of $\mathbf{A}^i \mathbf{s}$ (though it would not be much more difficult to take random vectors).

It should be noted that Wiedemann’s algorithm appeared previously in a quantum context in the algorithms of [13, 28] for multivariate quadratic equation systems. These algorithms construct large sparse matrices (Macaulay matrices) which need to be inverted. However, neither of these works considered a full reversible implementation of Wiedemann’s algorithm; instead they used generic reversibilisation results. In [28] they noticed that the algorithm could be implemented in a naive way, increasing the space complexity. In [13] they used the generic Bennett-Tompa trade-off [7] which introduces subexponential factors in the complexity (which disappear in the asymptotic complexity estimates).

In order to get a satisfying space complexity for the quantum Prange’s algorithm, we will need to implement Wiedemann’s algorithm in a reversible *and space-efficient* way. This is the goal of Sect. 4.

3 Quantum Preliminaries

In this section we give the required preliminaries of quantum computing and quantum ISD, including the formulation of our main result (Theorem 3) which relies on all the building blocks studied in the remainder of the paper.

We refer to [41] for an introduction to the notions of quantum computing (quantum states, amplitudes, ket notation $|\cdot\rangle$). We describe quantum algorithms in the *quantum circuit model* as a sequence of *quantum gates* applied to a set of qubits. We stand only at the *logical* level, in which gates can be freely applied to any qubit or pair of qubits without inducing any error. *Ancilla qubits* are those which start in the state $|0\rangle$ and are restored to $|0\rangle$ after applying the circuit.

Many advanced quantum cryptanalysis algorithms make use of quantum-accessible memories (also known as qRAM) [19,34]. qRAM can be seen as an abstraction of quantum hardware in which writing and/or reading in quantum superposition from a large-scale memory could be an efficient operation. It is notably used in certain quantum walk algorithms which require to maintain and update a large superposed memory state. However, it is almost certain that near-term quantum devices will not benefit from such capabilities. Going back to the *baseline* quantum circuit model, in which only fixed-arity gates can be used, these advanced algorithms lose their advantage. We refer to the aforementioned papers for more details on the qRAM model.

In this paper, we are interested in making conservative hardware assumptions, in which qRAM is not available and the number of logical qubits is limited. The new circuits that we design are entirely classical reversible circuits, which contain only NOT (X), CNOT (controlled-X, or CX) and Toffoli (double-controlled X, or CCX) gates. The other quantum gates required to run quantum ISD algorithms are Hadamard gates (H) used in the iterations of Quantum Amplitude Amplification, and rotation gates used in the construction of Dicke states [4] (which we define later on).

From an implementation perspective, CCX gates are known to be much more costly than X and CX gates, which is why they form the main target for optimization (e.g., in [30]).

3.1 Quantum Search

Grover’s quantum search algorithm [31] provides a quadratic speedup for any exhaustive search problem, which can be defined as the search of a preimage of 1 of a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, where $\{0, 1\}^n$ is the search space and f distinguishes “good” elements ($f(x) = 1$) from “bad” ones ($f(x) = 0$).

Prange’s information set decoding algorithm can be rephrased as such a search problem, which is why one can use quantum search here [10]. However, for this precise context it is better to rely on the generalization of Grover’s search known as Quantum Amplitude Amplification (QAA) [16]. QAA can start from any probabilistic algorithm (implemented as a quantum circuit) that succeeds with probability p , and needs $\mathcal{O}(1/\sqrt{p})$ iterations. Furthermore, it is quite robust if the probability of success is not known exactly, and p is only a lower bound. In that case an adapted procedure still succeeds in time $\mathcal{O}(1/\sqrt{p})$.

For the specific case of quantum ISD, we apply QAA in the following way.

Theorem 2 (Consequence of Theorem 2 and Theorem 3 of [16]). *Let U be a quantum circuit that, on input $|0\rangle$, produces a uniform superposition of N*

basis states (a subset $X \subseteq \{0, 1\}^n$):

$$U|0\rangle = \frac{1}{\sqrt{N}} \sum_{x \in X} |x\rangle \quad (4)$$

Let O_f be a quantum circuit that realizes a phase oracle for a function $f : X \mapsto \{0, 1\}$, where $|f^{-1}(1)| = M$:

$$\forall x \in X, O_f|x\rangle = (-1)^{f(x)}|x\rangle \quad (5)$$

Then, there exists a quantum algorithm that outputs an $x \in f^{-1}(1)$, and makes an expected number of $\Theta(\sqrt{N/M})$ calls to O_f and U .

More precisely, QAA is a procedure that runs with a fixed number of iterations, which repeat the *setup* operation U and the *test* operation O_f . A QAA iteration is the unitary $Q = -UO_0U^{-1}O_f$, where U and O_f are defined in Theorem 2, and O_0 is the unitary defined by $O_0|x\rangle = (-1)^{\delta_{0x}}|x\rangle$, where δ_{0x} is the Kronecker delta. That is, it flips the phase iff $x = 0$. One should note that O_0 is essentially an n -bit multi-controlled Z gate, which is equivalent (up to a Hadamard transform) to a multi-controlled Toffoli gate, which can be implemented with $\mathcal{O}(n)$ Toffoli gates [41]. As soon as U and/or O_f use more than $\mathcal{O}(n)$ depth, qubits and gates, the cost of this operation becomes negligible.

Let $\theta := \arcsin \sqrt{\frac{M}{N}}$. It can be shown [16] that starting from $U|0\rangle$ and applying k iterations of QAA, one produces the state:

$$Q^k U|0\rangle = \left(\frac{\sin((2k+1)\theta)}{\sqrt{M}} \sum_{x \in f^{-1}(1)} |x\rangle \right) + \left(\frac{\cos((2k+1)\theta)}{\sqrt{N-M}} \sum_{x \in f^{-1}(0)} |x\rangle \right) \quad (6)$$

This is why, knowing M (hence θ) in advance, we can succeed with probability close to 1 by setting $k = \lfloor \frac{\pi}{4\theta} \rfloor$. If we have only upper and lower bounds on M , we can use the following lemma.

Lemma 2. Assume that $M_\ell \leq M \leq M_u$. Run $k = \lfloor \frac{\pi}{4 \arcsin(\sqrt{M_u/N})} - \frac{1}{2} \rfloor$ iterations of QAA. The probability of success is:

$$p_{\text{succ}} \geq \sin^2 \left(\frac{\pi \arcsin \sqrt{M_\ell/N}}{2 \arcsin \sqrt{M_u/N}} - 2 \arcsin \sqrt{M_\ell/N} \right) \quad (7)$$

Proof. The choice of k ensures that $(2k+1) \arcsin \sqrt{\frac{M}{N}} \leq (2k+1) \arcsin \sqrt{\frac{M_u}{N}} \leq \frac{\pi}{2}$, which means the sin remains an increasing function. We can use Eq. 6 to bound the probability of measuring a good x :

$$\begin{aligned} p_{\text{succ}} &:= \sin^2 \left((2k+1) \arcsin \sqrt{\frac{M}{N}} \right) \geq \sin^2 \left((2k+1) \arcsin \sqrt{\frac{M_\ell}{N}} \right) \\ &\geq \sin^2 \left(\left(\frac{\pi}{2 \arcsin(\sqrt{M_u/N})} - 2 \right) \arcsin \sqrt{\frac{M_\ell}{N}} \right) \quad \square \end{aligned}$$

In particular, if M_ℓ and M_u are very close to M , then the success probability will become negligibly close to 1 (as it is when M is known exactly). If they are close up to a constant factor, then we ensure a constant probability of success.

3.2 Quantum ISD

Bernstein [10] noticed that Algorithm 1 is an exhaustive search for which one can use Grover’s algorithm. Quantum ISD was subsequently improved in [34], but not in a way that can be useful for us, since we refrain from using quantum RAM and exponential space.

Adapting Prange’s algorithm to the QAA framework is easily done, by defining the operators U and O_f :

- U produces a uniform superposition of subsets $I \subseteq \{0, \dots, n-1\}$ of size $n-k$:

$$|\mathcal{I}\rangle = \frac{1}{\sqrt{|S_{n,k}|}} \sum_{I \in S_{n,k}} |I\rangle, \tag{8}$$

where I is simply represented as a bit-vector of length n , where “1” in position i indicates that $i \in I$.

Such a quantum state is known in the literature as a *Dicke state*, and several efficient methods exist to compute it [4]. The cost of these methods is always significantly smaller than the cost of linear algebra in O_f (see, e.g., [46]).

- O_f is an oracle for the function f that takes as input I , and returns 1 if and only if \mathbf{H}_I is invertible and $\mathbf{H}_I^{-1}\mathbf{s}$ is of Hamming weight w .

Bernstein estimated that the evaluation of f would require $\mathcal{O}(n^3)$ “bit operations” [10]. This analysis was refined by further works [46]. However, to date, all implementations of O_f start by writing the matrix \mathbf{H}_I , then inverting it using Gaussian elimination. This strategy obviously requires at least $(n-k)^2$ qubits.

Decoding One out of Many. It is known that ISD algorithms can gain a speedup if the adversary’s goal is to decode a single syndrome out of many (the so-called DOOM problem [50]). A specific case of DOOM happens in the quasi-cyclic variant of the problem, which is used in BIKE [3] and HQC [1]. Indeed, in that case, one has $n = 2k$ and the parity-check matrix \mathbf{H} is formed of two circulant blocks \mathbf{H}_1 and \mathbf{H}_2 . If $\mathbf{e} = (\mathbf{e}_1\mathbf{e}_2)$ is the error vector, one has $(\mathbf{H}_1\mathbf{H}_2)(\mathbf{e}_1\mathbf{e}_2) = \mathbf{s}$. Let R_i be the rotation of a vector’s coordinates by i positions left, then due to the circulant structure of the blocks, one has:

$$\forall 0 \leq i < k, (\mathbf{H}_1\mathbf{H}_2)(R_i(\mathbf{e}_1)R_i(\mathbf{e}_2)) = R_i(\mathbf{s}). \tag{9}$$

As a consequence, \mathbf{e} can be found by decoding *any* of the k syndromes $R_i(\mathbf{s})$. One can then adapt Prange’s algorithm as follows. When a set I has been chosen, instead of computing $\mathbf{H}_I^{-1}\mathbf{s}$ and checking the Hamming weight of the result, one forms the $k \times k$ -dimensional matrix whose i -th column is $R_i(\mathbf{s})$, computes $\mathbf{H}_I^{-1}(R_0(\mathbf{s}) \cdots R_{k-1}(\mathbf{s}))$ and looks for a column of weight w . This increases the

probability of success by a factor k , and reduces the number of iterates in quantum Prange by a factor \sqrt{k} . This use of DOOM was discussed in [46].

Intriguingly, we do not know how to use DOOM in the context of Wiedemann inversion, because Wiedemann inverts the matrix on a single given vector. Doing this for another vector essentially requires to re-run the whole algorithm, without any gain. Therefore we will lose a factor \sqrt{k} in time complexity compared to [46] for the cases of BIKE and HQC.

3.3 Quantum Prange Using Wiedemann Inversion

We give a very abstract formulation of our main result, where the matrix is only accessed via a black-box representation of I and \mathbf{H}_J . In particular, this allows to consider alternative ways to represent the selection of a subset of columns.

From now on, we let \mathcal{J} be a set of bit-strings of fixed size such that there exists a surjective mapping F from \mathcal{J} to $S_{n,k}$, and furthermore, each subset has the same number of preimages. Consequently, sampling uniformly at random from \mathcal{J} allows to sample uniformly at random from $S_{n,k}$, even though \mathcal{J} may be a bigger set. Furthermore, given any $J \in \mathcal{J}$, we can extend our notation for sub-matrices by writing \mathbf{H}_J instead of $\mathbf{H}_{F(J)}$.

For our main result, we need a stronger heuristic than Heuristic 1, which indicates that being a solution, being an invertible matrix, and being a matrix on which Algorithm 2 succeeds with two iterations, are roughly independent events.

Heuristic 2. The proportion of matrices \mathbf{H}_J where I is a solution, which are invertible, and which Algorithm 2 can invert with $k = 2$ on input \mathbf{s} , is at least $0.288 \times 2^{-1.70} \simeq 2^{-3.50}$.

We now assume that we have the following:

- A quantum circuit Init that, on input $|0\rangle$, returns $|\mathcal{J}\rangle = \frac{1}{\sqrt{|\mathcal{J}|}} \sum_{J \in \mathcal{J}} |J\rangle$
- A quantum circuit $\text{Mult}_{\mathbf{H}}$ that, on input $|J\rangle |\mathbf{x}\rangle |\mathbf{y}\rangle$, where $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^{n-k}$, returns $|J\rangle |\mathbf{y} + \mathbf{H}_J \mathbf{x}\rangle |\mathbf{x}\rangle$

By their “space” complexity, we shall mean their entire qubit count, including ancillas. Our main result, building upon our implementation of Wiedemann’s algorithm in the quantum setting, integrates these two components in quantum ISD.

Theorem 3. *Given a circuit for Init with $S(\text{Init})$ qubits and $G(\text{Init})$ gates, and a circuit for $\text{Mult}_{\mathbf{H}}$ with $S(\text{Mult}_{\mathbf{H}})$ qubits and $G(\text{Mult}_{\mathbf{H}})$ gates, under Heuristic 2, there exists a quantum algorithm that solves the SDP with constant probability, using space: $\max(S(\text{Init}), S(\text{Mult}_{\mathbf{H}}) + \mathcal{O}(n))$, and gates:*

$$\mathcal{O} \left(\sqrt{\frac{\binom{n}{w}}{\binom{n-k}{w}}} \times (G(\text{Init}) + (n - k)G(\text{Mult}_{\mathbf{H}}) + (n - k)^2) \right) . \quad (10)$$

Proof. The algorithm is simply an adaptation of quantum Prange using QAA. Formally, our goal is not exactly to recover a subset I that yields the error vector \mathbf{e} , but a representation of it through J .

The operator U is simply `Init`. For the operator O_f , we use our quantum implementation of Wiedemann’s algorithm (Lemma 3), which has gate count $\mathcal{O}((n - k)G(\text{Mult}_{\mathbf{H}}) + (n - k)^2)$ and uses $S(\text{Mult}_{\mathbf{H}}) + \mathcal{O}(n)$ space. Importantly, in case of failure in Wiedemann’s algorithm, f will return 0. In case of success, we obtain the vector \mathbf{x} such that $\mathbf{H}_J \mathbf{x} = \mathbf{s}$. It remains to test if its Hamming weight is equal to w : the cost of this step is negligible with respect to the other components of the algorithm.

The number of iterations to perform depends on the probability that a given $J \in \mathcal{J}$ satisfies f , i.e., that the corresponding subset is solution, that \mathbf{H}_J is an invertible matrix, and that Wiedemann’s algorithm with two iterations succeeds. Under Heuristic 2, this probability can be lower bounded by: $2^{-3.50} \frac{\binom{n-k}{w}}{\binom{n}{w}}$. The result follows from Lemma 2. □

3.4 Quantum Circuit Components

In order to implement the Berlekamp-Massey and Wiedemann’s algorithms in an efficient and reversible manner, we need quantum circuits for several basic operations. These circuits are folklore and/or simple and/or borrowed from previous works; they are constructed entirely from X, CX and CCX gates.

We summarize here the main results needed, and the interested reader can find more details in the full version of the paper [20].

Fan-in. A *fan-in* circuit implements the operation:

$$|v_0, \dots, v_{n-1}, b\rangle \mapsto |v_0, \dots, v_{n-1}, b + (\sum_i v_i)\rangle .$$

It can be done using $\mathcal{O}(n)$ CX gates and in depth $\mathcal{O}(\log n)$.

Fan-out. A *fan-out* circuit implements the operation: $|b, 0, \dots, 0\rangle \mapsto |b, b, \dots, b\rangle$. It can be done using n CX gates and depth $\mathcal{O}(\log n)$.

Controlled-shift. A *controlled-shift* by a constant k maps:

$$\begin{cases} |0, v_0, \dots, v_{n-1}\rangle \mapsto |0, v_0, \dots, v_{n-1}\rangle \\ |1, v_0, \dots, v_{n-1}\rangle \mapsto |1, v_k, \dots, v_{n-1}, v_0, \dots, v_{k-1}\rangle \end{cases} \quad (11)$$

The shift is controlled by the first qubit. It can be done using $4n$ CX gates, $3n$ CCX gates, $\mathcal{O}(n)$ ancilla qubits and $\mathcal{O}(\log n)$ depth.

Reversion. A *reversion* circuit maps a register of n bits of the form:

$$1, b_0, \dots, b_{d-1}, 1, 0, \dots, 0 \quad \text{to:} \quad 1, b_{d-1}, \dots, b_0, 1, 0, \dots, 0 ,$$

i.e., it reverts the order of bits without taking into account the trailing zeroes (d being a variable), and assuming that the first bit is 1. It can be done using $\mathcal{O}(n \log n)$ gates, depth $\mathcal{O}(n)$ and using $\mathcal{O}(n)$ ancilla qubits.

(Constant) Matrix-vector Multiplication. The multiplication of an n -bit vector $\mathbf{x} \in \mathbb{F}_2^n$ by a *constant* matrix $\mathbf{H} \in \mathbb{F}_2^{m \times n}$:

$$|\mathbf{x}\rangle |\mathbf{y}\rangle \xrightarrow{\text{MultConstant}_{\mathbf{H}}} |\mathbf{x}\rangle |\mathbf{y} + \mathbf{H}\mathbf{x}\rangle ,$$

can be implemented using $\leq mn$ CX gates (the exact number depends on the matrix \mathbf{H}), depth $\max(m, n)$ and no ancilla qubits.

Circulant Matrix-vector Multiplication. In Sect. 5.3 we will use a quantum circuit for multiplication of a vector by a constant circulant matrix: we borrow its principle from Gidney [29].

When \mathbf{H} is a circulant matrix of dimension $n \times n$, there exists an implementation for $\text{MultConstant}_{\mathbf{H}}$ using $\mathcal{O}(n^{\log_2 3})$ CX gates, depth $\mathcal{O}(n^{\log_2 3})$ and $\mathcal{O}(n)$ ancilla qubits.

4 Space-Optimized Reversible Wiedemann Inversion

In this section, we detail our reversible implementation of Wiedemann’s matrix inversion, assuming that both the representation of column subsets (via the set \mathcal{J}) and the matrix-vector product are given as black-boxes, i.e., quantum circuits:

$$\begin{cases} \text{Init} : |0\rangle \mapsto \frac{1}{\sqrt{|\mathcal{J}|}} \sum_{J \in \mathcal{J}} |J\rangle \\ \text{Mult}_{\mathbf{H}} : |J\rangle |\mathbf{x}\rangle |\mathbf{y}\rangle \mapsto |J\rangle |\mathbf{y} + \mathbf{H}_J \mathbf{x}\rangle |\mathbf{x}\rangle . \end{cases} \quad (12)$$

We emphasize that the implementation of both components is not trivial, and that the time and space complexities of the iteration in Grover’s search depend in majority on them. But these implementations are deferred to Sect. 5.

The algorithm that we implement in this section is Algorithm 3, which corresponds to Wiedemann’s algorithm with two loop iterations. Notice that in case we succeed in the first iteration, we will have $\mathbf{t} = 0$ at Step 8, so the output value \mathbf{y} will remain unchanged. Therefore, whether the first or second loop iteration succeeds, the algorithm succeeds. Otherwise, even if the matrix is actually invertible, the Boolean flag `Success` will be set to `False`. This entire section proves the following result.

Lemma 3. *There exists a (classical) reversible circuit implementation of Algorithm 3 which uses $S(\text{Mult}_{\mathbf{H}}) + \mathcal{O}(n)$ qubits and $\mathcal{O}((n - k)G(\text{Mult}_{\mathbf{H}}) + (n - k)^2)$ gates.*

Proof. The remainder of this section proves that all steps of this algorithm can be implemented reversibly and efficiently.

- The computation of each sequence $(\mathbf{u}^T \mathbf{H}_J^i \mathbf{t})$ is done with Lemma 4.
- The evaluation of polynomials is done with Lemma 5.
- The Berlekamp-Massey algorithm for a sequence of length $\mathcal{O}(n - k)$ can be implemented with $\mathcal{O}((n - k)^2)$ gates and $\mathcal{O}((n - k) \log(n - k))$ depth by Lemma 6.

All these individual steps occupy a total of $\mathcal{O}(n)$ space for their outputs, which is erased by uncomputing them backwards once we have obtained the result. \square

Algorithm 3. Wiedemann's algorithm, simplified.

Constant: matrix \mathbf{H} , \mathbf{s} **Input:** J **Output:** a Boolean Success, and if Success = True, a vector \mathbf{x} such that $\mathbf{H}_J \mathbf{x} = \mathbf{s}$

- 1: $\mathbf{t} \leftarrow \mathbf{s}, \mathbf{y} \leftarrow 0, \mathbf{u} \leftarrow (1, 0, \dots, 0)$
 - 2: Compute the sequence $S = (\mathbf{u}^T \mathbf{H}_J^i \mathbf{t})_{0 \leq i \leq 2(n-k)}$ ▷ See Lemma 4
 - 3: Compute the minimal polynomial $C(X)$ of the sequence using the Berlekamp-Massey algorithm ▷ See Algorithm 5
 - 4: Let $C'(X) = (C(X) + 1)/X$
 - 5: $\mathbf{y} \leftarrow \mathbf{y} + C'(\mathbf{H}_J) \mathbf{t}$ ▷ See Lemma 5
 - 6: $\mathbf{t} \leftarrow \mathbf{t} + \mathbf{H}_J \mathbf{y}$ ▷ If success at the first step, here $\mathbf{t} = 0$
 - 7: $\mathbf{u} \leftarrow (0, 1, 0, \dots, 0)$
 - 8: Compute the sequence $S = (\mathbf{u}^T \mathbf{H}_J^i \mathbf{t})_{0 \leq i \leq 2(n-k)}$
 - 9: Compute the minimal polynomial $C(X)$ of the sequence
 - 10: Compute $C'(X) = (C(X) + 1)/X$
 - 11: $\mathbf{y} \leftarrow \mathbf{y} + C'(\mathbf{H}_J) \mathbf{t}$
 - 12: If $\mathbf{H}_J \mathbf{y} = \mathbf{s}$, then set Success to True (False otherwise)
 - 13: **Return** Success, \mathbf{y}
-

4.1 Evaluation of Matrix Powers

We elaborate here on a sequence of orthogonal polynomials in $\mathbb{F}_2[X]$. These polynomials arise from the fact that our $\text{Mult}_{\mathbf{H}}$ circuit, i.e., our matrix multiplication, is performed *out of place*.

Indeed, if a matrix \mathbf{A} is invertible, the operation $\mathbf{y} \mapsto \mathbf{A}\mathbf{y}$ is reversible. This means that there exists a reversible circuit performing the computation *in-place*: $|\mathbf{y}\rangle \mapsto |\mathbf{A}\mathbf{y}\rangle$. However, the reverse of this circuit would implement $|\mathbf{y}\rangle \mapsto |\mathbf{A}^{-1}\mathbf{y}\rangle$. This means that if we knew how to multiply by \mathbf{A} in place, we would essentially also know how to invert \mathbf{A} .

This is the reason why we start from an *out-of-place* matrix-vector multiplication, which is much easier to implement: $|\mathbf{x}\rangle |\mathbf{y}\rangle \xrightarrow{\text{Mult}_{\mathbf{A}}} |\mathbf{y} + \mathbf{A}\mathbf{x}\rangle |\mathbf{x}\rangle$, where \mathbf{x}, \mathbf{y} are two vectors. This is essentially a round of a Feistel scheme. It is easy to notice that $\text{Mult}_{\mathbf{A}}$ is a self-inverse operation followed by a swap, so it is reversible.

Unfortunately, Wiedemann's algorithm requires to compute iterations $\mathbf{A}^i \mathbf{x}$ given a starting vector \mathbf{x} . Since we are computing $\text{Mult}_{\mathbf{A}}$ out of place, naively computing a sequence of length $\mathcal{O}(n)$ would have us store $\mathcal{O}(n)$ intermediate vectors. We can do better than this through a family of orthogonal polynomials, which arise naturally by iterating $\text{Mult}_{\mathbf{A}}$.

Polynomials. We define the following polynomials:

$$\begin{cases} P_{-1}(X) = 0, P_0(X) = 1 \\ \forall i \geq 1, P_i(X) = P_{i-2}(X) + X P_{i-1}(X) \end{cases} \quad (13)$$

Then, the circuit resulting of iterating i times $\text{Mult}_{\mathbf{A}}$, noted $\text{Mult}_{\mathbf{A}}^i$, is such that:

$$\forall i \geq 0, \text{Mult}_{\mathbf{A}}^i |\mathbf{t}, \mathbf{0}\rangle = |P_i(\mathbf{A})\mathbf{t}, P_{i-1}(\mathbf{A})\mathbf{t}\rangle . \quad (14)$$

The proof is an elementary induction over i . Indeed, for all i :

$$\begin{aligned} \text{Mult}_{\mathbf{A}}^{i+1} |\mathbf{t}, \mathbf{0}\rangle &= \text{Mult}_{\mathbf{A}}(\text{Mult}_{\mathbf{A}}^i |\mathbf{t}, \mathbf{0}\rangle) = \text{Mult}_{\mathbf{A}} |P_{i+1}(\mathbf{A})\mathbf{t}, P_i(\mathbf{A})\mathbf{t}\rangle \\ &= |P_i(\mathbf{A})\mathbf{t} + \mathbf{A}P_{i+1}(\mathbf{A})\mathbf{t}, P_{i+1}(\mathbf{A})\mathbf{t}\rangle = |P_{i+2}(\mathbf{A})\mathbf{t}, P_{i+1}(\mathbf{A})\mathbf{t}\rangle . \end{aligned} \quad (15)$$

It can be noticed that for all $i \geq 0$, P_i is of degree i . As a consequence, there exists a binary, lower triangular (invertible) matrix M_ℓ such that:

$$M_\ell (P_0(X), P_1(X), \dots, P_\ell(X))^T = (1, X, \dots, X^\ell)^T . \quad (16)$$

We can now explain how to perform two important steps in Wiedemann's algorithm:

- Evaluating a sequence $u^T \mathbf{A}^i \mathbf{t}$
- Evaluating $C(\mathbf{A})\mathbf{t}$ for a polynomial C

both reversibly, and using only linear additional space.

Lemma 4. *Let \mathbf{A} be a matrix of dimension $n - k$. Given an implementation of $\text{Mult}_{\mathbf{A}}$ with G gates, depth D and $\mathcal{O}(n - k)$ space, there exists a reversible circuit to compute the sequence $u^T \mathbf{A}^i \mathbf{t}$ for $i = 0, \dots, \ell$ using $\mathcal{O}(\ell G + \ell^2)$ gates, depth $\mathcal{O}(\ell D + \ell \log \ell)$ and $\mathcal{O}(n - k + \ell)$ space.*

Proof. The idea is the following: we compute the $u^T \mathbf{A}^i \mathbf{t}$ as:

$$\left(u^T \mathbf{A}^0 \mathbf{t}, u^T \mathbf{A}^1 \mathbf{t}, \dots, u^T \mathbf{A}^\ell \mathbf{t} \right)^T = M_\ell \left(u^T P_0(\mathbf{A})\mathbf{t}, u^T P_1(\mathbf{A})\mathbf{t}, \dots, u^T P_\ell(\mathbf{A})\mathbf{t} \right)^T . \quad (17)$$

So, we only maintain two $(n - k)$ -qubit registers for computing the successive $P_i(\mathbf{A})\mathbf{t}$ in place, and ℓ qubits for the sequence. Each time we compute a new $P_i(\mathbf{A})\mathbf{t}$, we compute $u^T P_i(\mathbf{A})\mathbf{t}$ and then XOR it to the appropriate registers of the sequence. This operation requires a fan-out of depth $\mathcal{O}(\log \ell)$, which accounts for the additional depth $\ell \log \ell$.

Overall there will be $\mathcal{O}(\ell^2)$ CX operations performed. The complexity is dominated by the $\text{Mult}_{\mathbf{A}}$ operations. Once we have constructed the entire sequence, we perform the $\text{Mult}_{\mathbf{A}}$ s in reverse to erase the intermediate registers. \square

Lemma 5. *Let \mathbf{A} be a matrix of dimension $n - k$. Given an implementation of $\text{Mult}_{\mathbf{A}}$ with G gates, depth D and $\mathcal{O}(n - k)$ space, there exists a reversible circuit to compute $C(\mathbf{A})\mathbf{t}$ on an input polynomial $C(X)$ of degree $\leq \ell$ using $\mathcal{O}(\ell G + ((n - k) + \ell)\ell)$ gates, depth $\mathcal{O}(\ell D + \ell \log(n - k))$ and $\mathcal{O}(n - k + \ell)$ space.*

Algorithm 4. Classical Berlekamp-Massey algorithm

```

1: Input: sequence  $s_0, \dots, s_{N-1}$  in  $\mathbb{F}_q$ 
2: Output: retroaction polynomial  $C(X) \in \mathbb{F}_q[X]$ 
3:  $C(X) \leftarrow 1, B(X) \leftarrow 1$ 
4:  $L \leftarrow 0; m \leftarrow 1; b \leftarrow 1$ 
5: for all  $k = 1 \dots N - 1$  do
6:    $d \leftarrow s_k + \sum_{i=1}^L c_i s_{k-i}$ 
7:   if  $d = 0$  then ▷ Case 1
8:      $m \leftarrow m + 1$ 
9:   else if  $2L \leq k$  then ▷ Case 2
10:     $B(X), C(X) \leftarrow C(X), C(X) - \frac{d}{b} X^m B(X)$ 
11:     $L \leftarrow k + 1 - L; b \leftarrow d; m \leftarrow 1$ 
12:   else ▷ Case 3
13:     $C(X) \leftarrow C(X) - \frac{d}{b} X^m B(X)$ 
14:     $m \leftarrow m + 1$ 
15:   end if
16: end for
17: Return  $\text{Reversed}(C(X))$ 

```

Proof. The technique is very similar, using the fact that each $\mathbf{A}^i \mathbf{t}$ is a linear combination of the $P_j(\mathbf{A}) \mathbf{t}$ with fixed coefficients.

Let us write $M_\ell = (m_{ij})_{0 \leq i, j \leq \ell}$ and $C(X) = \sum_{i=0}^\ell c_i X^i$, then:

$$C(\mathbf{A}) \mathbf{t} = \sum_{i=0}^\ell c_i \mathbf{A}^i \mathbf{t} = \sum_{i=0}^\ell \sum_{j=0}^\ell m_{ij} c_i P_j(\mathbf{A}) \mathbf{t} = \sum_{j=0}^\ell \left(\sum_{i=0}^\ell m_{ij} c_i \right) P_j(\mathbf{A}) \mathbf{t} . \quad (18)$$

We start by computing the vector of all $c'_j := \left(\sum_{i=0}^\ell m_{ij} c_i \right)$ for $0 \leq j \leq \ell$, and storing this in $\ell + 1$ qubits. Afterwards we compute the sequence of the $P_j(\mathbf{A}) \mathbf{t}$, and depending on the stored coefficients, add this to our output register. The additional depth $\ell \log(n - k)$ comes from having to fan-out the current coefficient to control the addition to the output. □

4.2 Reversible Berlekamp-Massey

In order to explain our reversible implementation, we recall the Berlekamp-Massey algorithm [8] in Algorithm 4. We use N do denote the length of the input sequence, which will be $\mathcal{O}(n - k)$ in our case.

Similar to the reversible version of Euclide’s algorithm [49], we run a sequence of iterations where each one creates only $\mathcal{O}(1)$ bits of garbage, which can be stored. In our case, there are two such Boolean values: d , and a value v which decides if we enter case 2 or case 3 (leading to a modification of the polynomials, and of L).

First of all, since we focus on the binary case, the coefficient b is always 1 in the algorithm. Second, we notice that we can remove the variable m , by performing instead the operation $B(X) \leftarrow XB(X)$ each time we would have

Algorithm 5. Reversible Berlekamp-Massey algorithm for \mathbb{F}_2 .

```

1: Input: sequence  $s_0, \dots, s_{N-1}$  in  $\mathbb{F}_2$ 
2: Output: retroaction polynomial  $C(X) \in \mathbb{F}_2[X]$ 
3: Storage: register for  $C(X)$  ( $N$  bits),  $B(X)$  ( $N$  bits),  $L$  ( $N$  bits, in unary representation)
4: Garbage: register for  $d_1, \dots, d_N$ ,  $d_0 := 1$ , register for  $v_0, \dots, v_{N-1}$ 
5:  $C(X) \leftarrow 1, B(X) \leftarrow 1$ 
6:  $L \leftarrow 0$ 
7: for all  $k = 0 \dots N - 1$  do
8:    $d_k \leftarrow s_k + \sum_{i=1}^L c_i s_{k-i}$ 
9:    $v_k \leftarrow (2L \leq k)$  ▷ Boolean value deciding between case 2 and case 3
10:   $B(X) \leftarrow XB(X)$ 
11:  Conditioned on  $d_k = 1$  do
12:     $C(X) \leftarrow C(X) + B(X)$ 
13:  EndConditioned
14:  Conditioned on  $d_k v_k = 1$  do ▷ Remaining operations of case 2
15:     $B(X) \leftarrow B(X) + C(X)$ 
16:     $L \leftarrow k + 1 - L$  ▷ Can be done in place ( $k + 1$  is a constant here)
17:  EndConditioned
18: end for
19: Return  $\text{Reversed}(C(X))$ 

```

incremented m . This turns the algorithm into a less efficient version, but more suitable for reversibility.

Finally, we reorder the operations in the loop, as we notice that the shift $B(X) \leftarrow XB(X)$ is performed in all cases, and the operation $C(X) \leftarrow C(X) + B(X)$ is performed in all cases where $d = 1$. We obtain Algorithm 5.

Lemma 6. *Algorithm 5 can be implemented as a quantum circuit using $\mathcal{O}(N^2)$ quantum gates, $\mathcal{O}(N)$ space and depth $\mathcal{O}(N \log N)$.*

Proof. First of all, it is clear that each of the N loop iterations applies reversibly on the registers C, B, L, d_i, v_i

After performing these iterations, we copy the output $C(X)$. Then we compute the reverse of the iterations to erase all intermediate registers. After, we still need to reverse the polynomial $C(X)$. This is done in place using the implementation described in the full version of the paper [20].

In order to simplify the implementation, L is represented in *unary*, i.e., as a list of N bits (ℓ_1, \dots, ℓ_N) where $\ell_i = 1 \iff L \leq i$. This allows to perform the computation of d_k in $\mathcal{O}(N)$ gates (we perform the sum from $i = 1$ to N but use Toffoli gates with ℓ_i as inputs). The depth is $\mathcal{O}(\log N)$ using a fan-in circuit.

Then, v_k can be computed with $\mathcal{O}(1)$ operations since we can just access $\ell_{\lfloor k/2 \rfloor}$.

The shift of B can be implemented by swaps (which are not counted in the total number of gates, as they simply amount to renumbering the qubits). The two conditional XORs costs $\mathcal{O}(N)$ gates and depth $\mathcal{O}(\log N)$, needing again fan-outs of the control. In order to update the unary representation of L , we only

need $\mathcal{O}(N)$ gates, as we will apply X gates on the bits at positions before $k + 1$, then swap the entire sub-list (though k varies during the loop, it is a constant of the circuit). The depth is $\mathcal{O}(\log N)$, since this is also controlled and we need to fan-out the control.

Finally, the reversion of C costs $\mathcal{O}(N \log N)$ gates and depth $\mathcal{O}(N)$. We use no more than $\mathcal{O}(N)$ ancillas throughout the circuit. \square

4.3 Benchmarks

We denote by $Q = Q_J + 2(n - k) + A$ the total number of qubits used by the $\text{Mult}_{\mathbf{H}}$ circuit, where A is the number of ancilla qubits and Q_J the number of qubits used to represent J . We also denote by G_X, G_{CX} and G_{CCX} its respective X, CX and CCX gate counts.

Using our implementation of Berlekamp-Massey and Wiedemann's algorithms, we obtain the following counts. We neglect terms of smaller magnitude, except for the qubit count which is exact.

$$\left\{ \begin{array}{l} \text{Depth} \\ \text{Qubits} \\ \text{CCX Gates} \\ \text{CX Gates} \\ \text{X Gates} \end{array} \right. = \begin{array}{l} = 24D(n - k) + 152(n - k) \log_2(n - k) \\ = Q - A + 7(n - k + 1) + \max(A + 3(n - k) + 2, 10(n - k) + 11) \\ = 24(n - k)G_{CCX} + 116(n - k)^2 \\ = 24(n - k)G_{CX} + 356(n - k)^2 \\ = 24(n - k)G_X \end{array} \quad (19)$$

As our implementations of $\text{Mult}_{\mathbf{H}}$ will typically have quadratic gate count and depth at least linear in $(n - k)$, we can observe that this cost quickly dominates over the rest of the algorithm, though the additional terms are not negligible. The constant factors are also quite large, owing to the number of polynomial sequences evaluated during Wiedemann's algorithm and their size (twice $n - k$ to ensure success in the Berlekamp-Massey algorithm).

5 Implementing the Multiplication Circuit

In this section, we implement the multiplication circuit (with an implicit matrix). We propose two main approaches, using different representations of the choice of sub-matrix, i.e., different definitions of the set \mathcal{J} .

The first one (Sect. 5.1) is the approach chosen in previous works [46], where \mathcal{J} is the set of n -bit strings of Hamming weight $n - k$. In that case, Init is a unitary creating a so-called *Dicke state*, whose implementation can be borrowed from these previous works. Using this representation, we are able to decrease the space complexity of Wiedemann's algorithm (hence, the entire quantum Prange) to $\mathcal{O}(n)$.

The second one (Sect. 5.2) is based on permutations and sorting. In this approach, \mathcal{J} maps to a set of permutations of $\{0, \dots, n - 1\}$. Each permutation π of $\{0, \dots, n - 1\}$ naturally specifies a subset $I = \{\pi(0), \dots, \pi(n - k - 1)\} \in$

$S_{n,k}$. To the best of our knowledge, this idea has appeared in [45] but was not completely exploited. Our result shows a remarkable trade-off between qubit and gate count, where the qubit count increases to $\mathcal{O}(n \log^2 n)$, but remains comparable in practice to the space-efficient approach; while the total gate count remains at $\mathcal{O}(n^3)$, the constant factor is reduced, and the CCX gate count becomes asymptotically lower.

Using this second approach, further optimizations are possible (Sect. 5.3), although they do not perform well for practical parameters at the moment.

5.1 Space-Optimized Circuits

In this subsection, \mathcal{J} is the set of n -bit strings of Hamming weight $n - k$, which is identified with $S_{n,k}$.

Lemma 7. *There exists a reversible circuit implementing the $\text{Mult}_{\mathbf{H}}$ operation:*

$$|J\rangle |\mathbf{x}\rangle |\mathbf{y}\rangle \xrightarrow{\text{Mult}_{\mathbf{H}}} |J\rangle |\mathbf{y} + \mathbf{H}_J \mathbf{x}\rangle |\mathbf{x}\rangle \tag{20}$$

which uses $\mathcal{O}(n)$ space, $\mathcal{O}(n(n - k))$ gates and depth $\mathcal{O}(n \log(n - k))$.

Proof. The operation that we implement is basically the computation of $\mathbf{H}_J \mathbf{x}$, except that we will directly XOR the result to \mathbf{y} .

Let $\mathbf{x} := (x_0, \dots, x_{n-k-1})$. Furthermore, let $\mathbf{a} = (a_0, a_1, \dots, a_{n-k-1})$ be a vector of integers where a_0 is the position of the first “1” in J , a_1 the second one, etc. We note the coefficients of \mathbf{H} as (h_{ij}) and $\mathbf{H}_J = (h'_{ij})$, then by definition of a_j :

$$\forall 0 \leq i \leq n - k - 1, \forall 0 \leq j \leq n - k - 1, h'_{ij} = h_{ia_j} = \bigoplus_{k=0}^{n-1} \delta_{a_j k} h_{ik} . \tag{21}$$

Thus, we can express $\mathbf{H}_J \mathbf{x}$ as follows:

$$\begin{aligned} \forall 0 \leq i \leq n - k - 1, (\mathbf{H}_J \mathbf{x})_i &= \bigoplus_{j=0}^{n-k-1} h'_{ij} x_j = \bigoplus_{j=0}^{n-k-1} \bigoplus_{\ell=0}^{n-1} \delta_{a_j \ell} h_{i\ell} x_j \\ &= \bigoplus_{\ell=0}^{n-1} \left(\bigoplus_{j=0}^{n-k-1} \delta_{a_j \ell} x_j \right) h_{i\ell} . \end{aligned} \tag{22}$$

Our strategy is to compute the vector $\mathbf{v} := \left(\bigoplus_{j=0}^{n-k-1} \delta_{a_j \ell} x_j \right)_{0 \leq \ell \leq n-1}$. This vector simply places the coordinates of x at the positions marked by J , keeping their order. As an example, if we have $J = (0, 1, 0, 0, 1, 1, \dots)$, then \mathbf{v} will start with $(0, x_0, 0, 0, x_1, x_2, \dots)$.

In order to do so, we maintain a unary counter \mathbf{e} , implemented as a register with $n - k$ bits, which remains of weight 1, and represents the number c such that $\mathbf{e}_c = 1$, i.e., such that the bit of weight c in \mathbf{e} equals 1.

For $\ell = 0$ to $n - 1$, we compute: $v_\ell = j_\ell \mathbf{e} \cdot \mathbf{x}$ where j_ℓ is the ℓ th-bit in the register J . Indeed, the dot-product $\mathbf{e} \cdot \mathbf{x}$ selects a new coordinate in \mathbf{x} each time the counter is updated. Then, we perform a shift of \mathbf{e} , controlled on j_ℓ , to update the counter c as $c \leftarrow c + j_\ell$. These operations require $\mathcal{O}(n - k)$ CCX gates and $\mathcal{O}(\log(n - k))$ depth (due to the use of fan-in and fan-out circuits).

Once we have computed v_ℓ , we use another fan-out and update the output $\mathbf{H}_J \mathbf{x}$. Indeed, from Eq. 22 we have:

$$\forall 0 \leq i \leq n - k - 1, (\mathbf{H}_J x)_i = \bigoplus_{\ell=0}^{n-1} v_\ell h_{i\ell} . \quad (23)$$

So we simply need to XOR v_ℓ at the right positions. This costs $\mathcal{O}(n - k)$ CXs. We then uncompute the fan-out, erase v_ℓ and go to the next iteration. Since there are n iterations, the overall gate count and depth are respectively $\mathcal{O}(n(n - k))$ and $\mathcal{O}(n \log(n - k))$. \square

Cost Formulas. We computed asymptotic formulas for this space-optimized $\text{Mult}_{\mathbf{H}}$ circuit (left), and combined them with Eq. 19 to obtain the cost of the inversion circuit (right):

$$\left\{ \begin{array}{l} \text{Depth} = 4n \log_2(n - k) \\ \text{Qubits} = n + 6(n - k) + 2 \\ \text{CCX Gates} = 5n(n - k) \\ \text{CX Gates} = 9n(n - k) \\ \text{X Gates} = 2 \end{array} \right. \quad \left\{ \begin{array}{l} \text{Depth} = 96n(n - k) \log_2(n - k) \\ \text{Qubits} = n + 19(n - k) + 18 \\ \text{CCX Gates} = 120n(n - k)^2 \\ \text{CX Gates} = 216n(n - k)^2 \\ \text{X Gates} = \mathcal{O}(n - k) \end{array} \right. \quad (24)$$

The multiplication of constants between the $\text{Mult}_{\mathbf{H}}$ circuit and the inversion circuit creates even larger constants, which are far from negligible for actual parameters.

5.2 Toffoli-Optimized Circuits

In this second approach, the set \mathcal{J} is defined by means of a *sorting network*. Note that [46] used similar tools to permute the columns of the matrix \mathbf{H} within the QAA iteration; our reasoning is different here since we directly implement $\text{Mult}_{\mathbf{H}}$.

A sorting network with n entries is defined as a sequence of *comparators* and *switches*, which respectively compare a pair of entries at fixed positions, and swap them depending on the result of the comparison. While sorting networks with $\mathcal{O}(n \log n)$ comparators exist [2], one of the most efficient in practice is Batcher's odd-even mergesort [5], which has depth $\lceil \log_2 n \rceil (\lceil \log_2 n \rceil + 1)/2$ and contains $\frac{n}{4} \lceil \log_2 n \rceil (\lceil \log_2 n \rceil - 1) + n - 1$ comparators. This is the one we use here.

Let (A_0, \dots, A_{n-1}) be an n -tuple of integers. Let us define the mapping \mathcal{N} from (A_0, \dots, A_{n-1}) to bit-strings of length $\frac{n}{4} \lceil \log_2 n \rceil (\lceil \log_2 n \rceil - 1) + n - 1 = \mathcal{O}(n \log^2 n)$ which gives the results of all comparisons in the sorting network,

where the comparators are taken in a fixed, arbitrary order. While the sorting network itself is not reversible, storing $\mathcal{N}(A_0, \dots, A_{n-1})$ is sufficient to make it reversible. This increases the space usage of Batcher’s network to $\mathcal{O}(n \log^2 n)$.

Definition of Init. Equipped with the mapping \mathcal{N} above, we now define \mathcal{J} as:

$$\mathcal{J} = \left\{ \left((A_0, \dots, A_{n-1}), \mathcal{N}(A_0, \dots, A_{n-1}) \right), A_0, \dots, A_{n-1} \in [0; n^3 - 1] \right\} . \quad (25)$$

That is, we take an n -tuple of integers (A_0, \dots, A_{n-1}) between 0 and n^3 , append the result of all comparisons in the network, and identify this as a bit-string.

The unitary `Init`, which creates the uniform superposition over \mathcal{J} , essentially consists in taking uniform superposition of such integers (which is efficient) and computing a reversible sorting network. In particular, the bit-string $\mathcal{N}(A_0, \dots, A_{n-1})$ is computed only once, at this step, and used later in the multiplication circuit without the need to recompute it. As a comparison of integers can be performed without ancillas using a modified CDKM addition circuit [22], `Init` uses almost no ancillas.

Besides removing the need for Dicke states, this definition will make the multiplication circuit less costly, as we show later.

Mapping to a Subset of Columns. We explain here how an element $J \in \mathcal{J}$ defines a subset $S_{n,k}$, and why all subsets have the same probability to appear. This mapping is especially important for the definition of `MultH`.

First of all, an element $J \in \mathcal{J}$ defines a permutation π_J of $\{0, \dots, n - 1\}$, which is the permutation such that sorting A_0, \dots, A_{n-1} puts the integer A_i in position $\pi_J(i)$. This permutation can easily be implemented by a *switching network*. This network has the same structure as the sorting network that defines \mathcal{N} , but it is made only of controlled swaps (the switches), which are controlled by the bits of $\mathcal{N}(A_0, \dots, A_{n-1})$.

It is well-known that, if we sort n *distinct* entries chosen uniformly at random, the permutation π_J is also uniformly random. By choosing entries with sufficiently many bits, they will all be distinct with large probability.

Lemma 8. *Let (A_0, \dots, A_{n-1}) be drawn uniformly at random in $[0; n^3 - 1]$, then they are all distinct with probability at least $1 - \frac{1}{2n}$.*

Proof. We simply lower bound the probability of all A_i to be distinct, as:

$$\left(1 - \frac{1}{n^3} \right) \left(1 - \frac{2}{n^3} \right) \cdots \left(1 - \frac{n - 1}{n^3} \right) \geq 1 - \sum_{i=1}^{n-1} \frac{i}{n^3} \geq 1 - \frac{1}{2n} . \quad \square$$

In the case where the entries are not distinct, we do not know if the algorithm will succeed. Luckily, our implementation of Wiedemann’s inversion ensures that there are no false positives, so we can still use QAA (Theorem 3). Indeed,

we know that the oracle f returns 1 for the tuples (A_0, \dots, A_{n-1}) for which all the numbers are distinct, and the corresponding permutation returns a solution, so the probability of success of the amplified algorithm is at least $(1 - \frac{1}{2n}) 2^{-3.50 \frac{\binom{n-k}{w}}{\binom{n}{w}}}$.

Finally, the permutation π_J defines a subset of columns from $S_{n,k}$ as follows: the positions of the columns are $\pi_J(0), \pi_J(1), \dots, \pi_J(n-k-1)$. As π_J is a uniformly random permutation (when selecting J at random from \mathcal{J}), the subset $\{\pi_J(0), \pi_J(1), \dots, \pi_J(n-k-1)\}$ is also a uniformly random element of $S_{n,k}$.

Definition of the Multiplication Circuit. We make a small tweak to the definition of the sub-matrix \mathbf{H}_J . Since we defined a permutation of columns, it makes sense to define \mathbf{H}_J as:

$$(\mathbf{H}_J)_{ij} := (h_{i\pi_J(j)}) \quad (26)$$

This definition is slightly different from the one of Sect. 5.1, where the columns were put in a fixed order. Here, the columns of \mathbf{H}_J will also be permuted. This has no incidence on the rest of the algorithm.

We can now implement our circuit for $\text{Mult}_{\mathbf{H}}$, which takes as input an element of \mathcal{J} . In fact, this circuit does not need the integers (A_0, \dots, A_{n-1}) , which we are keeping along only for the sake of reversibility. It only relies on the bit-string $\mathcal{N}(A_0, \dots, A_{n-1})$ which defines the permutation π_J .

Lemma 9. *There exists a reversible circuit implementing the $\text{Mult}_{\mathbf{H}}$ operation:*

$$|J\rangle |\mathbf{x}\rangle |\mathbf{y}\rangle \xrightarrow{\text{Mult}_{\mathbf{H}}} |J\rangle |\mathbf{y} + \mathbf{H}_J \mathbf{x}\rangle |\mathbf{x}\rangle \quad (27)$$

using $\mathcal{O}(n \log^2 n)$ space, $\mathcal{O}(n \log^2 n)$ CCX gates, $\mathcal{O}(n^2)$ CX gates and depth $n + o(n)$.

Proof. The idea of the circuit is very similar to Lemma 7. First, we compute the vector \mathbf{v} that places the input bits \mathbf{x} at appropriate positions, i.e., bit x_i in position $\pi_J(i)$. Then, we compute the fixed matrix-vector product $\mathbf{H}\mathbf{v}$.

The first step is done using the switching network, i.e., a series of $\mathcal{O}(n \log^2 n)$ controlled swaps with depth $\mathcal{O}(\log^2 n)$.

The second step can be done in depth n and $\mathcal{O}(n^2)$ CX gates as recalled in Sect. 3.4 (see the full version [20] for more details). \square

Interestingly, the dominating operation becomes the product of \mathbf{v} by the constant matrix \mathbf{H} . This is a linear quantum circuit, which can be implemented with only CX gates. The depth is also asymptotically optimal. This appears clearly on our asymptotic cost formulas for this alternative function:

$$\begin{cases} \text{Depth} & = n \\ \text{Qubits} & = n + (n - k) + \frac{n}{4} \lceil \log_2 n \rceil (\lceil \log_2 n \rceil - 1) + n - 1 + 3n \log_2 n \\ \text{CCX Gates} & = \frac{1}{2} n (\log_2 n)^2 \\ \text{CX Gates} & = n(n - k) \\ \text{X Gates} & = \mathcal{O}(1) \end{cases} \quad (28)$$

Having much lower constants than Eq. 24, these counts yield much more favorable results when we plug them in Eq. 19:

$$\begin{cases} \text{Depth} & = 24n(n - k) \\ \text{Qubits} & = \frac{n}{4} \lceil \log_2 n \rceil (\lceil \log_2 n \rceil - 1) + n + 19(n - k) + 17 + 3n \log_2 n \\ \text{CCX Gates} & = 12n(n - k)(\log_2 n)^2 + 116(n - k)^2 \\ \text{CX Gates} & = 24n(n - k)^2 + 356(n - k)^2 \\ \text{X Gates} & = \mathcal{O}(n - k) \end{cases} \quad (29)$$

In both these formulas, the term $3n \log_2 n$ in the qubit count comes from the initial tuple of integers (A_0, \dots, A_{n-1}) . They actually do not intervene in the definition of the circuits, but we need to keep them along in order to be able to invert the Init circuit. This term is asymptotically negligible, but not entirely when $n \simeq 10^3$.

5.3 Gate-Optimized Multiplication Circuit for Circulant Matrices

In the case of BIKE [3] and HQC [1], one has $n = 2k$ and the parity-check matrix \mathbf{H} is made of two $k \times k$ circulant blocks. Therefore, we can replace the multiplication by \mathbf{H} by a more efficient circuit using Karatsuba multiplication of polynomials (detailed in the full version of the paper [20]). While our benchmarks show a noticeable improvement in total gate count, the downside is an increase in depth, since the Karatsuba circuit that we use, based on Gidney [29], has asymptotically worse depth.

Asymptotically, binary polynomial multiplication can be performed in $\tilde{\mathcal{O}}(n)$ binary operations, for example using Cantor's algorithm [18] in $\mathcal{O}(n(\log n)^{1.585})$. This means that there exists a circuit for multiplication by a circulant matrix using $\mathcal{O}(n(\log n)^{1.585})$ gates and qubits, and consequently:

Theorem 4. *If the parity-check matrix is block-circulant, there exists a quantum algorithm solving SD for random codes using $\mathcal{O}\left(n^2(\log n)^2 \times \sqrt{\frac{\binom{n}{k}}{\binom{n-t}{k}}}\right)$ gates and $\mathcal{O}(n(\log n)^2)$ qubits.*

This decrease of the gate count is specific to our “sorting-based” approach, using the fact that \mathbf{H} is structured and that Wiedemann's algorithm can make use of this. To the best of our knowledge, this is the first asymptotic improvement over the $\mathcal{O}(n^3)$ linear algebra factor in quantum ISD to date.

Unfortunately, while efficient classical software exists [17], corresponding quantum circuits for circulant matrix-vector multiplication have not been studied as much. In particular, the constant factors, depth and qubit counts of this method remain unknown.

6 Evaluation of Costs for Code-Based Cryptosystems

In this section we give resource estimates for the three inversion circuits detailed in Sect. 5, and compare them.

6.1 Comparison of Circuits

We computed the number of gates, qubits and depth of our circuits for parameters of the three round 4 candidates for post-quantum key-exchange based on codes at the NIST post-quantum standardization: Classic McEliece [12], BIKE [3] and HQC [1]. We compare them with the counts of [46] in Table 1.

Even if we disregard the use of memory, it is difficult to compare our results with the advanced quantum ISD algorithms that could apply here [34–36], since they considered only asymptotic complexities and neglected polynomial factors. However, it is likely that these algorithms could benefit from improved linear algebra circuits.

We note that, while we compare here with [46], Bonnetain and Jaques also designed a quantum circuit for binary Gaussian elimination for a matrix of dimension $(n - k) \times (n - k)$ with depth $\mathcal{O}((n - k) \log(n - k))$ [14]. This is better than the depth $\mathcal{O}((n - k)^2)$ reported in [46], so we believe their counts could be immediately improved by using the circuit of [14] as a replacement. Nevertheless, our main focus in Table 1 is on the number of qubits.

Let us consider the Classic McEliece parameters for NIST security level 1, which are at least as secure as AES-128 against Grover’s exhaustive key search (“McEliece L1” in Table 1). Using the space-optimized circuit, the total number of qubits required for quantum Prange is $18 + n + 19(n - k) = 18\,098$, instead of $2^{22} \simeq 4\,194\,304$ reported in [46]. Previously one would have needed at least $(n - k)^2 = 589\,824$ qubits at best to store the matrix being inverted using Gaussian elimination. Our improvement brings the number of logical qubits to the same order as the one required in factoring large instances of RSA [30] via Shor’s algorithm [51].

However, this optimization of space comes at the expense of gate count and depth. Indeed, both increase a thousandfold, mostly due to the large constant factors appearing in Eq. 24. Overall, the product between depth and width of the circuit (so-called “DW” metric) increases slightly.

The sorting-based approach has a much better trade-off. On the same example, it will use 258 769 qubits, among which 115 104 are used to store the state of switches, and 125 568 to store the initial numbers which are sorted. This increase in the space complexity comes entirely from our representation of the column choice, which could likely be compacted. On this example, the total gate count and depth are significantly reduced but remain a factor 2^6 above those of [46]. The difference is more favorable for larger code lengths as the Toffoli count is asymptotically smaller.

With the same amount of qubits, the use of Karatsuba-based multiplication of polynomials for the matrix-vector product reduces the gate count asymptotically. The difference is already noticeable for the BIKE and HQC parameters. However, our implementation is not optimized in depth. As a consequence the DW product increases significantly.

Table 1. Quantum resource estimates for the QAA iteration. Counts are given in \log_2 and rounded. The number of CCX gates is not given in [46], but due to the structure of the Gaussian elimination circuit, it is of the same order as the total number of gates.

Implementation	Scheme	n	k	Counts (in \log_2)				
				CCX	Total gates	Depth	Qubits	DW
[46]	BIKE L1	24646	12323		43	28	29	57
	BIKE L3	49318	24659		46	31	31	62
	BIKE L5	81946	40973		48	32	33	65
	HQC L1	35338	17669		45	30	30	60
	HQC L3	71702	35851		47	32	32	64
	HQC L5	115274	57637		50	34	34	68
	McEliece L1	3488	2720		30	20	22	42
	McEliece L3	4608	3360		32	22	23	45
	McEliece L5-1	6688	5024		34	23	24	47
	McEliece L5-2	6960	5413		33	23	24	47
	McEliece L5-3	8192	6528		34	23	24	47
	Space-optimized Section 5.1	BIKE L1	24646	12323	48.7	50.2	38.9	18.0
BIKE L3		49318	24659	51.7	53.2	41.0	19.0	60.0
BIKE L5		81946	40973	53.9	55.4	42.5	19.7	62.2
HQC L1		35338	17669	50.2	51.7	40.0	18.5	58.5
HQC L3		71702	35851	53.3	54.8	42.1	19.5	61.7
HQC L5		115274	57637	55.4	56.8	43.5	20.2	63.7
McEliece L1		3488	2720	37.8	39.3	31.7	14.1	45.8
McEliece L3		4608	3360	39.6	41.1	32.9	14.8	47.7
McEliece L5-1		6688	5024	41.0	42.5	33.9	15.2	49.1
McEliece L5-2		6960	5413	40.9	42.3	33.8	15.2	49.0
McEliece L5-3		8192	6528	41.3	42.8	34.1	15.3	49.4
Toffoli-optimized Section 5.2		BIKE L1	24646	12323	39.5	46.4	32.8	21.8
	BIKE L3	49318	24659	41.7	49.4	34.8	22.9	57.7
	BIKE L5	81946	40973	43.4	51.6	36.2	23.8	60.0
	HQC L1	35338	17669	40.8	47.9	33.8	22.4	56.3
	HQC L3	71702	35851	43.0	51.0	35.9	23.6	59.4
	HQC L5	115274	57637	44.3	53.0	37.2	24.3	61.5
	McEliece L1	3488	2720	32.0	35.9	26.2	18.5	44.7
	McEliece L3	4608	3360	33.4	37.6	27.2	19.1	46.3
	McEliece L5-1	6688	5024	34.3	38.9	28.1	19.6	47.8
	McEliece L5-2	6960	5413	34.3	38.8	28.1	19.7	47.8
	McEliece L5-3	8192	6528	34.6	39.2	28.4	19.9	48.3
	Karatsuba Section 5.3	BIKE L1	24646	12323	39.5	44.3	40.3	21.8
BIKE L3		49318	24659	41.7	46.8	42.9	23.0	65.8
BIKE L5		81946	40973	43.4	49.1	45.2	23.8	69.0
HQC L1		35338	17669	40.8	46.3	42.4	22.4	64.8
HQC L3		71702	35851	43.0	48.9	45.0	23.6	68.6
HQC L5		115274	57637	44.3	49.6	45.7	24.3	70.0

6.2 Discussion

Our work does not threaten the security of the NIST code-based candidates Classic McEliece, BIKE and HQC. In fact, it does not overall improve the circuit depth with respect to [46] and [14], and the gains in DW product that we

observed with respect to [46] come mostly from the reduction in qubits. Besides, we lose the gain of DOOM that is exploitable with Gaussian elimination in the case of BIKE and HQC, as mentioned in Sect. 3.2. However, DOOM reduces the number of iterations by a factor \sqrt{n} , while our method reduces the Toffoli gate count (and the total gate count for block-circulant matrices) by a factor of order $\frac{n}{\log^2 n}$, which is asymptotically better.

While our space-optimized circuit reaches quite competitive qubit counts, we have observed that the Toffoli-optimized approach offers a better trade-off in practice, and can be combined with an improved matrix multiplication circuit for block circulant matrices. There are several ways in which this approach can be further improved.

First of all, the bottleneck of the cost in Toffoli (CCX) gates is the switching network that is used in $\text{Mult}_{\mathbf{H}}$. Right now, this network contains $\mathcal{O}(n \log^2 n)$ controlled swaps. However, it is known that given a permutation π of $\{0, \dots, n-1\}$, one can design a network with only $\mathcal{O}(n \log n)$ swaps that implements π . Such an algorithm is described in detail in [11], but the difficulty would be to implement it as an efficient quantum circuit. We would use this circuit once in the QAA iteration and store the network using $\mathcal{O}(n \log n)$ qubits. The CCX gate of the $\text{Mult}_{\mathbf{H}}$ operation would further decrease to $\mathcal{O}(n \log n)$.

The bottleneck in the space complexity is the integers (A_0, \dots, A_{n-1}) which we use as intermediates to sample a random permutation, and the state of the comparators which we use to represent it. Other ways to generate a random permutation (e.g., the Fisher-Yates shuffle) did not seem competitive. However, our approach right now is quite conservative, as we ensured that the permutation was sampled uniformly at random. This requirement can be relaxed: we only want to sample from a family of permutations that distribute well the subset of $n-k$ columns to be selected, so that the probability of finding a solution remains high. It is known that switching networks with $\mathcal{O}(n \log n)$ and depth $\mathcal{O}(\log^2 n)$ with good mixing properties can be constructed [23]. We believe that such a construction could be used to reduce both the CCX gate count and number of qubits, but leave this as future work.

7 Conclusion

In this paper, we achieved new trade-offs in the linear algebra circuit required in the quantum Prange's algorithm. In particular, we can bring the number of qubits down to $\mathcal{O}(n)$, at a level similar to what Shor's algorithm requires for large RSA instances. The core idea is to use Wiedemann's matrix inversion algorithm, where the matrix to invert is only implicitly represented. Our main contribution is a complete reversible and space-efficient implementation of this algorithm with detailed gate counts.

While our new approach removes the limitation of the number of qubits, we still expect quantum ISD to remain unrealizable for code-based cryptosystems, even for large-scale quantum computers, due to its large circuit depth and gate count requirements.

Nevertheless, our result greatly improves the known time-memory trade-offs [27], and switches the focus towards the time complexity. In this context, we also showed that Wiedemann inversion, combined with an appropriate representation of column permutations in Prange’s algorithm, improves the Toffoli (CCX) gate count with respect to Gaussian elimination. It can also improve the overall gate count in the case of circulant matrices. Our estimations shows that these improvements are observable for actual parameters of code-based cryptosystems, but further dedicated circuit optimizations could significantly enhance these results.

Finally, although this paper focused on the quantum Prange algorithm, our implementation of Wiedemann’s algorithm is of independent interest, as there are other quantum algorithms that need to inverse a sparse or implicit matrix, for example solving multivariate polynomial equation systems [13, 28]. Our circuit could be used to reduce the memory complexity, and perhaps estimate more precisely the time complexity of such methods.

Acknowledgments.. We would like to thank the anonymous reviewers of ASIACRYPT 2024 for helpful remarks. This work has been supported by the French Agence Nationale de la Recherche through the CROWD project under Contract ANR-CE 48 2022, and through the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

References

1. Aguilar Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G., Bos, J., Dion, A., Lacan, J., Robert, J.M., Véron, P.: Hamming quasi-cyclic (HQC). Submission to the NIST PQC process, Round 4 (2022), <https://pqc-hqc.org/>
2. Ajtai, M., Komlós, J., Szemerédi, E.: An $O(n \log n)$ sorting network. In: STOC. pp. 1–9. ACM (1983). <https://doi.org/10.1145/800061.808726>
3. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Güneysu, T., Aguilar Melchor, C., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.P., Zémor, G., Vasseur, V., Ghosh, S., Richter-Brokmann, J.: BIKE: bit flipping key encapsulation. Submission to the NIST PQC process, Round 4 (2022), <https://bikesuite.org/>
4. Bärttschi, A., Eidenbenz, S.J.: Short-depth circuits for Dicke state preparation. In: QCE. pp. 87–96. IEEE (2022). <https://doi.org/10.1109/QCE53715.2022.00027>
5. Batcher, K.E.: Sorting networks and their applications. In: AFIPS Spring Joint Computing Conference. AFIPS Conference Proceedings, vol. 32, pp. 307–314. Thomson Book Company, Washington D.C. (1968). <https://doi.org/10.1145/1468075.1468121>
6. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1+1 = 0$ improves information set decoding. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7237, pp. 520–536. Springer (2012). https://doi.org/10.1007/978-3-642-29011-4_31
7. Bennett, C.H.: Time/space trade-offs for reversible computation. *SIAM J. Comput.* **18**(4), 766–776 (1989)

8. Berlekamp, E.R.: Algebraic coding theory. McGraw-Hill series in systems science, McGraw-Hill (1968), <https://www.worldcat.org/oclc/00256659>
9. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.A.: On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Inf. Theory* **24**(3), 384–386 (1978). <https://doi.org/10.1109/TIT.1978.1055873>
10. Bernstein, D.J.: Grover vs. mceliece. In: *PQCrypto. Lecture Notes in Computer Science*, vol. 6061, pp. 73–80. Springer (2010). https://doi.org/10.1007/978-3-642-12929-2_6
11. Bernstein, D.J.: Verified fast formulas for control bits for permutation networks. *IACR Cryptol. ePrint Arch.* p. 1493 (2020), <https://eprint.iacr.org/2020/1493>
12. Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., Wang, W.: Classic McEliece: conservative code-based cryptography. Submission to the NIST PQC process, Round 4 (2022), <https://classic.mceliece.org>
13. Bernstein, D.J., Yang, B.: Asymptotically faster quantum algorithms to solve multivariate quadratic equations. In: *PQCrypto. Lecture Notes in Computer Science*, vol. 10786, pp. 487–506. Springer (2018). https://doi.org/10.1007/978-3-319-79063-3_23
14. Bonnetain, X., Jaques, S.: Quantum period finding against symmetric primitives in practice. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**(1), 1–27 (2022). <https://doi.org/10.46586/TCHES.V2022.I1.1-27>
15. Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security. In: *PQCrypto. Lecture Notes in Computer Science*, vol. 10786, pp. 25–46. Springer (2018). https://doi.org/10.1007/978-3-319-79063-3_2
16. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. *Contemporary Mathematics* **305**, 53–74 (2002). <https://doi.org/10.1090/conm/305/05215>
17. Brent, R.P., Gaudry, P., Thomé, E., Zimmermann, P.: Faster multiplication in $\text{gf}(2)[x]$. In: *ANTS. Lecture Notes in Computer Science*, vol. 5011, pp. 153–166. Springer (2008). https://doi.org/10.1007/978-3-540-79456-1_10
18. Cantor, D.G.: On arithmetical algorithms over finite fields. *J. Comb. Theory, Ser. A* **50**(2), 285–300 (1989). [https://doi.org/10.1016/0097-3165\(89\)90020-4](https://doi.org/10.1016/0097-3165(89)90020-4)
19. Chailloux, A., Debris-Alazard, T., Etinski, S.: Classical and quantum algorithms for generic syndrome decoding problems and applications to the lee metric. In: *PQCrypto. Lecture Notes in Computer Science*, vol. 12841, pp. 44–62. Springer (2021). https://doi.org/10.1007/978-3-030-81293-5_3
20. Chevignard, C., Fouque, P., Schrottenloher, A.: Reducing the number of qubits in quantum information set decoding. *IACR Cryptol. ePrint Arch.* p. 907 (2024), <https://eprint.iacr.org/2024/907>
21. Cooper, C.: On the distribution of rank of a random matrix over a finite field. *Random Struct. Algorithms* **17**(3-4), 197–212 (2000)
22. Cuccaro, S.A., Draper, T.G., Kutin, S.A., Moulton, D.P.: A new quantum ripple-carry addition circuit (2004)
23. Czumaj, A.: Random permutations using switching networks. In: *STOC*. pp. 703–712. ACM (2015). <https://doi.org/10.1145/2746539.2746629>
24. Dornstetter, J.: On the equivalence between Berlekamp’s and Euclid’s algorithms (corresp.). *IEEE transactions on information theory* **33**(3), 428–431 (1987)
25. Ducas, L., Esser, A., Etinski, S., Kirshanova, E.: Asymptotics and improvements of sieving for codes. In: *EUROCRYPT (6). Lecture Notes in Computer Science*, vol. 14656, pp. 151–180. Springer (2024). https://doi.org/10.1007/978-3-031-58754-2_6

26. Esser, A., Ramos-Calderer, S., Bellini, E., Latorre, J.I., Manzano, M.: An optimized quantum implementation of ISD on scalable quantum resources. *IACR Cryptol. ePrint Arch.* p. 1608 (2021), <https://eprint.iacr.org/2021/1608>
27. Esser, A., Ramos-Calderer, S., Bellini, E., Latorre, J.I., Manzano, M.: Hybrid decoding - classical-quantum trade-offs for information set decoding. In: *PQCrypto. Lecture Notes in Computer Science*, vol. 13512, pp. 3–23. Springer (2022). https://doi.org/10.1007/978-3-031-17234-2_1
28. Faugère, J., Horan, K., Kahrobaei, D., Kaplan, M., Kashefi, E., Perret, L.: Fast quantum algorithm for solving multivariate quadratic equations. *CoRR abs/1712.07211* (2017), <http://arxiv.org/abs/1712.07211>
29. Gidney, C.: Asymptotically efficient quantum karatsuba multiplication. arXiv preprint [arXiv:1904.07356](https://arxiv.org/abs/1904.07356) (2019)
30. Gidney, C., Ekerå, M.: How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* **5**, 433 (2021). <https://doi.org/10.22331/Q-2021-04-15-433>, <https://doi.org/10.22331/q-2021-04-15-433>
31. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: *STOC*. pp. 212–219. ACM (1996). <https://doi.org/10.1145/237814.237866>
32. Guo, Q., Johansson, T., Nguyen, V.: A new sieving-style information-set decoding algorithm. *IACR Cryptol. ePrint Arch.* p. 247 (2023), <https://eprint.iacr.org/2023/247>
33. Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing grover oracles for quantum key search on AES and LowMC. In: *EUROCRYPT (2). Lecture Notes in Computer Science*, vol. 12106, pp. 280–310. Springer (2020). https://doi.org/10.1007/978-3-030-45724-2_10
34. Kachigar, G., Tillich, J.: Quantum information set decoding algorithms. In: *PQCrypto. Lecture Notes in Computer Science*, vol. 10346, pp. 69–89. Springer (2017). https://doi.org/10.1007/978-3-319-59879-6_5
35. Kimura, N., Takayasu, A., Takagi, T.: Memory-efficient quantum information set decoding algorithm. In: *ACISP. Lecture Notes in Computer Science*, vol. 13915, pp. 452–468. Springer (2023). https://doi.org/10.1007/978-3-031-35486-1_20
36. Kirshanova, E.: Improved quantum information set decoding. In: *PQCrypto. Lecture Notes in Computer Science*, vol. 10786, pp. 507–527. Springer (2018). https://doi.org/10.1007/978-3-319-79063-3_24
37. Lee, P.J., Brickell, E.F.: An observation on the security of McEliece’s public-key cryptosystem. In: *EUROCRYPT. Lecture Notes in Computer Science*, vol. 330, pp. 275–280. Springer (1988). https://doi.org/10.1007/3-540-45961-8_25
38. Massey, J.L.: Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory* **15**(1), 122–127 (1969). <https://doi.org/10.1109/TIT.1969.1054260>
39. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\mathcal{O}(2^{0.054n})$. In: *ASIACRYPT. Lecture Notes in Computer Science*, vol. 7073, pp. 107–124. Springer (2011). https://doi.org/10.1007/978-3-642-25385-0_6
40. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: *EUROCRYPT (1). Lecture Notes in Computer Science*, vol. 9056, pp. 203–228. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_9
41. Nielsen, M.A., Chuang, I.: *Quantum computation and quantum information* (2002)
42. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf>

43. NIST: Post-quantum cryptography: Digital signature schemes - round 1 additional signatures (2023), <https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures>
44. NIST: Round 4 standardisation results for the post-quantum cryptography standardization process (2024), <https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions>
45. Perriello, S.: Design and development of a quantum circuit to solve the information set decoding problem (2017)
46. Perriello, S., Barenghi, A., Pelosi, G.: Improving the efficiency of quantum circuits for information set decoding. *ACM Transactions on Quantum Computing* **4**(4), 1–40 (2023)
47. Prange, E.: The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory* **8**(5), 5–9 (1962). <https://doi.org/10.1109/TIT.1962.1057777>
48. Qiskit contributors: Qiskit: An open-source framework for quantum computing (2023). <https://doi.org/10.5281/zenodo.2573505>
49. Roetteler, M., Naehrig, M., Svore, K.M., Lauter, K.E.: Quantum resource estimates for computing elliptic curve discrete logarithms. In: ASIACRYPT (2). *Lecture Notes in Computer Science*, vol. 10625, pp. 241–270. Springer (2017). https://doi.org/10.1007/978-3-319-70697-9_9
50. Sendrier, N.: Decoding one out of many. In: PQCrypto. *Lecture Notes in Computer Science*, vol. 7071, pp. 51–67. Springer (2011). https://doi.org/10.1007/978-3-642-25405-5_4
51. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: FOCS. pp. 124–134. IEEE Computer Society (1994). <https://doi.org/10.1109/SFCS.1994.365700>
52. Stern, J.: A method for finding codewords of small weight. In: Coding Theory and Applications. *Lecture Notes in Computer Science*, vol. 388, pp. 106–113. Springer (1988). <https://doi.org/10.1007/BFB0019850>
53. Wiedemann, D.H.: Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory* **32**(1), 54–62 (1986). <https://doi.org/10.1109/TIT.1986.1057137>



On the Semidirect Discrete Logarithm Problem in Finite Groups

Christopher Battarbee^{1(✉)}, Giacomo Borin^{2,14(✉)}, Julian Brough¹³,
Ryann Cartor³, Tobias Hemmert¹³, Nadia Heninger⁴, David Jao⁵,
Delaram Kahrobaei^{6,7}, Laura Maddison⁸, Edoardo Persichetti⁹,
Angela Robinson¹⁰, Daniel Smith-Tone^{10,11}, and Rainer Steinwandt¹²

¹ Sorbonne University, CNRS, LIP6, PolSys, Paris, France
`christopher.battarbee@lip6.fr`

² IBM Research Europe, Rüschlikon, Switzerland
`sdlp@gbor.in`

³ Clemson University, Clemson, USA

⁴ University of California, San Diego, USA

⁵ University of Waterloo, Waterloo, ON, Canada

⁶ Department of Computer Science and Mathematics, Queens College,
City University of New York, Flushing, USA

⁷ Department of Computer Science and Engineering, Tandon School of Engineering,
New York University, New York, USA

⁸ University of Ottawa, Ottawa, ON, Canada

⁹ Florida Atlantic University, Boca Raton, USA

¹⁰ National Institute of Standards and Technology, Gaithersburg, USA

¹¹ University of Louisville, Louisville, USA

¹² University of Alabama in Huntsville, Huntsville, USA

¹³ Bundesamt für Sicherheit in der Informationstechnik, Bonn, Germany

¹⁴ University of Zurich, Zürich, Switzerland

Abstract. We present an efficient quantum algorithm for solving the semidirect discrete logarithm problem (SDLP) in *any* finite group. The believed hardness of the semidirect discrete logarithm problem underlies more than a decade of works constructing candidate post-quantum cryptographic algorithms from non-abelian groups. We use a series of reduction results to show that it suffices to consider SDLP in finite simple groups. We then apply the celebrated Classification of Finite Simple Groups to consider each family. The infinite families of finite simple groups admit, in a fairly general setting, linear algebraic attacks providing a reduction to the classical discrete logarithm problem. For the sporadic simple groups, we show that their inherent properties render them unsuitable for cryptographically hard SDLP instances, which we illustrate via a Baby-Step Giant-Step style attack against SDLP in the Monster Group.

Our quantum SDLP algorithm is fully constructive, up to the computation of maximal normal subgroups, for all but three remaining cases that appear to be gaps in the literature on constructive recognition of groups; for these cases SDLP is no harder than finding a linear represen-

tation. We conclude that SDLP is not a suitable post-quantum hardness assumption for any choice of finite group.

Keywords: Group-Based Cryptography · Semidirect Discrete Logarithm Problem · Post-Quantum Cryptography

1 Introduction

There has been a significant amount of research on *semidirect product* cryptography within the post-quantum community [23, 24, 28, 41, 42] since its introduction in 2013 by Habeeb et al. [24]. This approach aims to use the group-theoretic notion of the semidirect product to generalize the discrete logarithm problem (DLP) in a manner that resists quantum attacks. The resulting problem is called the *Semidirect Discrete Logarithm Problem* (SDLP), and is the subject of this paper.

The NIST Post-Quantum Standardization process [39] has motivated work on a wide variety of computational problems and candidate constructions for post-quantum cryptographic algorithms. While lattice-based cryptography may currently be the most well-represented among post-quantum schemes, there is a desire to have a diverse collection of candidates, computational hardness assumptions and algorithms. This would provide a hedge against cryptanalytic surprises (such as the late-breaking attacks against Rainbow and SIKE) and allow for different performance tradeoffs, as well as advanced functionalities.

In this light, SDLP is an appealing generalization of DLP over cyclic groups that can be used to define analogues of discrete logarithm-based cryptography over non-commutative (semi-)groups. SDLP offers an unusual degree of flexibility; almost all of the cryptosystems are defined for *any* finite group, and several are defined for finite semigroups. Battarbee et al. [6, 7] showed that the machinery of SDLP gives rise to a group action and suggests that this might allow efficiency improvements over other candidates for *group-action* based cryptography, especially in the realm of digital signature schemes.

Historically, cryptanalysis of SDLP-based schemes has been specific to a particular choice of group. For example, there have been several proposals of groups to be used with Semidirect Product Key Exchange (SDPKE), which is the analogue of Diffie-Hellman Key Exchange (DHKE) for SDLP [23, 24, 28, 41, 42]. Each of these proposals was later shown to be insecure due to some feature of the selected platform group [16, 36–38, 43]. However, analogously to the relationship between DHKE and the Diffie-Hellman problems, a break of SDPKE for some group does not demonstrate that SDLP is easy in that group. More recently, Imran and Ivanyos [25] showed that SDLP in a solvable group admits a reduction to standard quantum-vulnerable problems. While this work has eliminated some candidate constructions, it leaves unresolved the question motivating our work: is there any choice of finite group G such that SDLP in G is post-quantum secure?

This question has remained unanswered for over a decade of active research in the area. In this work, we prove that the answer is negative. Our result makes use of the famous Classification of Finite Simple Groups and develops

a generalization of the “decomposition” methods of [25]. In particular, we will repeatedly use the “recursion tool” of [25] to reduce an instance of SDLP in an arbitrary finite group to several instances of SDLP in finite simple groups. Since there is a relatively short and known list of all possible finite simple groups, we then devise quantum and classical algorithms for solving SDLP or reducing it to the problem of finding a linear representation of the group, that we can solve (up to some technical detail concerning constructive recognition of groups) in each family of finite simple groups.

Our contributions are highlighted below.

- We develop a more sophisticated method of decomposition into “smaller” instances of SDLP, based on the ideas of [25]. In particular we show that, for SDLP in an arbitrary finite group G , one can always generate logarithmically-many instances of SDLP in simple groups; moreover, solving these instances of SDLP suffices to solve SDLP in the group G .
- We solve SDLP in non-sporadic simple groups by studying their representations and, building on another idea of [25], give a reduction to the classical DLP after some linear algebra calculations of polylogarithmic complexity.
- We propose an adaptation of Shanks’ Baby-Step-Giant-Step algorithm which efficiently (and classically) solves SDLP in sporadic groups, exploiting the relatively low orders of their elements. This completes our claim that one can solve SDLP in a practical manner in an arbitrary finite group G .

While our work eliminates hope for quantum-secure SDLP-based cryptography over finite groups, the corresponding problem for semigroups, which is featured in some previous proposals [24], remains an interesting open problem. Indeed, evidence suggests that some group-theoretic problems may be harder to solve on semigroups than on groups. For example, Childs and Ivanyos [17] prove an exponential lower bound on the number of quantum queries required to solve the constructive semigroup membership problem on a black-box semigroup, whereas the corresponding problem for black-box groups is known to be quantum polynomial-time since it simply reduces to DLP. We remark also that our techniques are unlikely to translate to the infinite case of SDLP.

1.1 Paper Organization and Contributions

We prove the following main results.

Theorem 1. *Let G be a finite black-box group. Given an oracle computing maximal normal subgroups, in order to solve SDLP in G , it suffices to solve SDLP in at most $\log |G|$ many simple groups. We can compute the information defining these instances of SDLP in simple groups in quantum polynomial time in $\log |G|$.*

Theorem 2. *Let G be a finite black-box group and suppose there is an efficient linear (or projective) representation of G of dimension n . One can solve SDLP in G in quantum polynomial time in n and $\log |G|$.*

Corollary 1. *Let S be a finite simple black-box group, that is not one of the groups ${}^2F_4(2^{2n+1})$ or ${}^3D_4(2^e)$. One can solve SDLP in S in quantum polynomial time in $\log |S|$.*

We will explicitly discuss SDLP in the two groups omitted by Corollary 1 in Sect. 6. The techniques for computing arbitrary maximal normal subgroups comes from the literature on various computational group theoretic problems, in particular the task of computing composition series of groups. The literature here does not appear to be completely resolved, and we discuss it in Appendix A. The rest of our paper is organized as follows (which also gives a guide to the structure of our results). Section 2 gives some background on group theory and some of the computational problems that arise in this work. This section also summarizes the main results of [25] that we generalize in this work. In Sect. 3, we go into more detail on the main decomposition tool, and generalize it in several steps to finite simple groups. In Sect. 4, we give a generic method to solve SDLP for any finite group using its linear representation. Combining the results in these two sections gives an efficient reduction of SDLP in any group to SDLP in finite simple groups, as well as an algorithm solving SDLP with running time dependent on the faithful dimension in simple groups. In Sect. 5, we use the classification of finite simple groups to iterate through each of the families of finite simple groups in turn. Given the previous computational reductions, the main question for each of these families is to construct an efficient linear representation from a black-box group; this is known to be in probabilistic quantum polynomial time for all but two minor special cases. Finally, the sporadic groups can be easily dispensed with, either via a brute-force search or via an adapted baby-step giant-step algorithm. We conclude in Sect. 6 that SDLP on finite groups is not a reliable candidate for quantum-resistant cryptography.

2 Preliminaries

The semidirect discrete logarithm problem arises from the study of the semidirect product of a group G by its own automorphism group. Let us briefly recall the definition:

Definition 1 (Holomorph). *Let G be a group with automorphism group $\text{Aut}(G)$. The semidirect product of G by $\text{Aut}(G)$, written $G \rtimes \text{Aut}(G)$, is the set of ordered pairs from $G \times \text{Aut}(G)$ equipped with multiplication defined by*

$$(g, \phi)(g', \psi) := (g\phi(g'), \phi \circ \psi)$$

where \circ denotes function composition. We call this structure the holomorph of G and denote it by $\text{Hol}(G)$.

By induction, one can verify that for $(g, \phi) \in \text{Hol}(G)$ and $x \in \mathbb{N}$, we have

$$(g, \phi)^x = (\underbrace{g\phi(g) \dots \phi^{x-1}(g)}_{=: s_{g, \phi}(x)}, \phi^x),$$

and we can think of this as a function $s_{g,\phi} : \mathbb{Z} \rightarrow G$, mapping the exponent x to the projection onto the G -component of $(g, \phi)^x$. For finite groups G , the order of elements in $\text{Hol}(G)$ is bounded above by $|G|$ (see [11]), so we may, without loss of generality, choose to restrict the domain of $s_{g,\phi}$ to a finite set.

Definition 2 (Semidirect Discrete Logarithm Problem). *Let G be a group and fix $(g, \phi) \in \text{Hol}(G)$. Suppose $h = s_{g,\phi}(x)$ for some $x \in \mathbb{Z}$. We define $\text{SDLP}(G, \phi, g, h)$ to be the set consisting of all the integers i such that $s_{g,\phi}(i) = h$. The Semidirect Discrete Logarithm Problem (SDLP) is to determine this set.*

Remark 1. It will be useful in some contexts for us to say “SDLP for G and ϕ ”, for a finite group G and one of its automorphisms ϕ . By this, we just mean an instance of SDLP where one recovers $\text{SDLP}(G, \phi, g, h)$, without wishing to specify g and h .

Since $s_{g,\phi}(x)$ is the projection of a holomorph element onto one of its coordinates, the SDLP setup does not directly expose an element of G or $\text{Aut}(G)$. The problem is therefore not trivially equivalent to a standard DLP. Thinking of $s_{g,\phi}$ in terms of a projection also tells us how to efficiently compute it: we can compute exponentiation in the holomorph using standard square-and-multiply techniques, and then project the result to obtain the desired value.

2.1 Essential Group Theory Notions

Let G be a group. A subgroup $N \leq G$ is said to be *normal* if for all $g \in G$ and $n \in N$, $gn g^{-1} \in N$. We use $N \triangleleft G$ to denote that N is a normal subgroup of G . We can then define the *quotient group* G/N to be the set of left cosets of N in G . In other words, $G/N = \{gN \mid g \in G\}$. The group operation on G/N is induced by the group operation on G in the obvious way.

A group G is *simple* if it has no non-trivial proper normal subgroups, and we refer to a subgroup H of a group G as *characteristic* if $\phi(H) = H$ for every automorphism $\phi \in \text{Aut}(G)$. The group G is said to be *characteristically simple* if it has no non-trivial proper characteristic subgroups. The example $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ illustrates that being characteristically simple is a strictly weaker property than being simple. A subnormal series $1 = H_m \triangleleft H_{m-1} \triangleleft \dots \triangleleft H_1 \triangleleft H_0 = H$ of a group H is called a *composition series* if each quotient H_i/H_{i-1} is simple and called a *quasi-composition series* if each quotient is either abelian or non-abelian simple.

For technical reasons we require that any computational representation of a group G comes with two attributes `CS_Abelian_Flag`, and `CS_NonAbelianFlag`, which are by default set to 0 (i.e., $G.\text{CS_Abelian_Flag} = G.\text{CS_NonAbelian_Flag} = 0$). One of our algorithms later on may update these values if it detects that the group is either of two special cases of characteristically simple.

A *linear representation* of a group G on a finite-dimensional vector space V is a group homomorphism

$$\psi : G \rightarrow \text{GL}(V).$$

Here, $\text{GL}(V)$ denotes the general linear group on V . We also consider *projective* linear representations, i.e., homomorphisms $G \rightarrow \mathbb{PGL}(V)$, where $\mathbb{PGL}(V) \cong$

$\text{GL}(V)/Z(\text{GL}(V))$ contains the invertible linear maps acting on $\mathbb{P}(V)$ (since scalar matrices act trivially on $\mathbb{P}(V)$). If $\mathbf{A} \in \text{GL}(V)$ we write $[\mathbf{A}]$ for the corresponding class in $\mathbb{P}\text{GL}(V)$.

Black-Box Groups. The introduction of *black-box groups* can be traced back to Babai and Szemerédi [4] as a useful abstraction of computations in groups.

Definition 3 (Black-Box Group). *A black-box group $G \subset \{0, 1\}^n$ is a group whose elements are bit strings of length n , endowed with an oracle that performs the group operations, multiplication and inversion, and can check if one element is the identity or not (this is equivalent to checking if two elements are equal or not).*

As an additional requirement, for technical reasons we will need our black-box groups to come equipped with a unique labelling; that is, a function λ on the bitstrings representing the group that is such that $\lambda(x) = \lambda(y)$ if and only if x and y represent the same group element.

The use of black-box oracles for groups is not new to cryptography. As an example, Shoup proved lower bounds for generic algorithms solving DLP using black-box groups [47]. This is a conservative computational model for cryptanalysis of SDLP-based cryptography, since any construction instantiated on a particular group will need to be able to perform operations on the base group G (and $\text{Aut}(G)$) and test the equality of the resulting operations.

The Black-Box Group model is also of interest for computational group theorists as a tool to investigate the complexity of several group related problems such as the Hidden-Subgroup Problem [26], or in relation to “The computational matrix group project” [34, 40].

Of particular relevance is the **Constructive Recognition Problem**, proposed by Babai and Beals [1, Section 9.2], in which one is asked to find a computationally efficient isomorphism between a simple black-box group and an explicitly defined simple group. Observe that for the case of cyclic groups of prime order this problem reduces exactly to DLP since, given $\phi : G \xrightarrow{\sim} \mathbb{Z}/p\mathbb{Z}$, we can easily compute logarithms (divisions) in $\mathbb{Z}/p\mathbb{Z}$.

Several works [1, 2, 12, 13, 27, 29, 30] have investigated the constructive recognition problem for other families of simple groups; this is commonly done by reducing it to the case of $\mathbb{P}\text{SL}(2, q)$ using so-called *number theory oracles*, i.e., oracles for solving discrete logarithm and factoring, to handle large finite-field computations [2, 18]. These algorithms thus run in quantum polynomial time [46].

2.2 Related Work and Known Results

Broadly speaking, there are two main categories of literature on SDLP: cryptographic constructions based on the Semidirect Product Key Exchange (SDPKE) and the associated cryptanalysis, and algorithmic analysis of the underlying SDLP problem itself.

The first category of literature encompasses a decades-long cat-and-mouse game between papers suggesting parameters and choices of groups to instantiate SDPKE [23, 24, 28, 41, 42], and works cryptanalyzing the results [16, 36–38, 43]. These papers occur as responses to each other, in the sense that new proposals are patches to avoid the attacks of prior works. For a detailed review of the chronology see [8].

In the same way that the security of DHKE is not precisely equivalent to DLP, the security of SDPKE is not precisely equivalent to SDLP. The works mentioned above do not address the complexity of solving SDLP; the first result in this direction dates to 2022. This and subsequent such results form the second category of literature mentioned above, which also includes the present paper. Battarbee et al. [6] pointed out a connection to group actions and later exploited it [7] to give a subexponential quantum algorithm for SDLP.

Mendelsohn et al. [35] found faster methods for some small parameters. Most recently, Imran and Ivanyos [25] gave an efficient polynomial-time quantum algorithm to solve SDLP for solvable groups and matrix groups with certain associated endomorphisms. Our work is a generalization of this paper to all finite groups.

Imran and Ivanyos introduce two important notions, which we sketch here. The first is that, given a group G and a normal subgroup N , in order to solve SDLP in G , it suffices to solve SDLP in N and G/N . The second is that, if G is a matrix group, we can show that SDLP reduces to an instance of DLP after the application of some linear algebraic methods.¹ Suppose we can compute a composition series of an arbitrary group G ; then, provided the composition factors are suitable matrix groups (or elementary abelian groups, in which SDLP is predictably easy), we can use the decomposition algorithm inductively to solve SDLP in the composition factors and to recover a solution of SDLP in the group that we started in. This breaks, among other things, all the finite solvable groups (which includes every group proposed for use with SDLP-based cryptography).

Our work can be seen as a more sophisticated version of this method. By refining the method of computing the appropriate subgroups we can reduce the solution to solving appropriate instances of SDLP in the simple groups. In addition, we construct a generalization of the reduction in a matrix group that turns out to be particularly effective for simple groups. Indeed, because we know that only the simple groups listed by the classification of simple groups can appear in this decomposition, and since we can show that each of these is vulnerable to some method of solving SDLP, we can show that SDLP is easy for any finite group, resolving a loose conjecture of [25].

For the purpose of describing our algorithms let us recall some of the known results relating to the structure of SDLP.

Prior Results. One of the main ideas of [25] is to reframe SDLP as an orbit problem. For each pair (g, ϕ) in the holomorph of G consider the function $\rho_{(g, \phi)}$

¹ Interestingly, this method is somewhat similar to the “linear decomposition” attacks presented in the analysis of SDPKE.

defined by $\rho_{(g,\phi)}(h) = g\phi(h)$. It is not difficult to check by induction that $\rho_{(g,\phi)}^x(h) = g\phi(g)\cdots\phi^{x-1}(g)\phi^x(h)$. We therefore get the following equivalent definition of SDLP.

Definition 4 (SDLP(G, ϕ, g, h)). Let G be a finite group, and $\phi \in \text{Aut}(G)$ be one of its automorphisms. Suppose $h = \rho_{(g,\phi)}^x(1_G)$ for some $x \in \mathbb{N}$. We define the set $\text{SDLP}(G, \phi, g, h)$ to be the set of integers i satisfying

$$h = \rho_{(g,\phi)}^i(1_G).$$

The Semidirect Discrete Logarithm Problem, or SDLP, is to determine this set.

We will use both variants interchangeably. Let us also recall some of the results on the set of solutions to SDLP: the following is a synthesis of ideas found in [6,7]. In the following, the symbol 1 refers to the integer value 1, and 1_G denotes the group identity; these are (clearly) not the same.

Theorem 3. Let G be a finite group and ϕ one of its automorphisms. Consider SDLP for $g, h \in G$. There exists an integer n_0 (dependent on g and ϕ) such that $\rho_{g,\phi}^{n_0}(1_G) = s_{g,\phi}(n_0) = 1_G$, and the set

$$\{1_G, s_{(g,\phi)}(1), \dots, s_{(g,\phi)}(n_0 - 1)\} = \{1_G, \rho_{(g,\phi)}(1_G), \dots, \rho_{(g,\phi)}^{n_0-1}(1_G)\}$$

has size n_0 , and is exactly the codomain of $s_{(g,\phi)}$. We have that one can compute n_0 in quantum polynomial time with a Shor-like period-finding algorithm, and that the solution set $\text{SDLP}(G, \phi, g, h)$ is of the form

$$\{t_0 + tn_0 : t \in \mathbb{Z}\}$$

where $0 \leq t_0 < n_0$.

Finally, although some of the ideas of [25] are given in detail in the main body of the present paper, we will just quote the fact given as [25, Theorem 6] that one can solve SDLP in an elementary abelian group in time polynomial in the input size of the group. This will be necessary since several of the results on simple groups will require that the simple group is non-abelian, and finite cyclic groups of prime order are the only abelian simple groups. Note also that, although our more general ideas capture the result of [25] for solving SDLP in solvable groups, their specific methods may be slightly more efficient in practice for this particular case.

3 The Main Reduction

Recall from the discussion in the previous section that Imran and Ivanyos [25] provide a solution for SDLP in solvable groups by descending a composition series (using Theorem 3 in their paper), at each step encountering an easy variant of SDLP in an elementary abelian group. In this section, we significantly generalize the results of [25], by using their method to completely reduce an arbitrary instance of SDLP to several instances of SDLP in a simple group. In particular, Theorem 6 demonstrates that if we know how to compute maximal normal subgroups, in order to solve some instance of SDLP in a finite group G , it suffices to solve at most $\log |G|$ instances of SDLP in a simple group. The data describing each of these instances of SDLP can be obtained in time quantum polynomial in $\log |G|$.

We will defer the proof of this result to the end of the section. We begin by developing more sophisticated techniques for computing the subgroups required for [25, Theorem 3], and devise a contingency for the case in which no such subgroups exist.

3.1 Reduction to SDLP in Simple Groups

Let us review the central “recursion tool” of Imran-Ivanyos [25, Theorem 3]. The main idea of the recursion tool is to demonstrate that if we can find a normal subgroup N of G that is invariant under our automorphism, solving SDLP(G, ϕ) can be reduced to solving SDLP($N, (\phi|_N)^{n_0}$) and SDLP($G/N, \bar{\phi}$) for some n_0 , automorphism $\bar{\phi}$ and suitable elements.

We will state and prove the result in full, in order to review ideas from its proof that are important in our reduction algorithms. For these purposes, we first provide the following lemma concerning powers of $\rho_{(g,\phi)}$.

Lemma 1. *Let $g \in G, \phi \in \text{Aut}(G)$. For any integer x , then $\rho_{(g,\phi)}^{-x}(h) := (\rho_{(g,\phi)}^x)^{-1}(h) = \rho_{(\phi^{-1}(g^{-1}), \phi^{-1})}^x(h)$. Additionally, for any $m, n \in \mathbb{Z}$, then $\rho_{(g,\phi)}^{mn} = \rho_{(\rho_{(g,\phi)}^n(1_G), \phi^n)}$.*

Proof. The first statement follows from the observation that $(\rho_{(g,\phi)})^{-1}(f) = \phi^{-1}(g^{-1}f) = \rho_{(\phi^{-1}(g^{-1}), \phi^{-1})}(h)$.

As $\rho_{(g,\phi)}^x(h) = \rho_{(g,\phi)}^x(1_G)\phi^x(h)$ it suffices to prove the statement for $h = 1_G$. Assume first m is positive. If n is also positive then

$$\rho_{(\rho_{(g,\phi)}^n(1_G), \phi^n)}^m(1_G) = \prod_{i=0}^{m-1} (\phi^n)^i \left(\rho_{(g,\phi)}^n(1_G) \right) = \rho_{g,\phi}^{mn}(1_G).$$

While for negative n applying the formula for $\rho_{(g,\phi)}^{-x}$ above yields

$$\rho_{(g,\phi)}^{mn}(1_G) = \rho_{(\phi^{-1}(g^{-1}), \phi^{-1})}^{m(-n)}(1_G) = \rho_{(\rho_{(\phi^{-1}(g^{-1}), \phi^{-1})}^{-n}(1_G), \phi^n)}^m(1_G) = \rho_{(\rho_{(g,\phi)}^n(1_G), \phi^n)}^m(1_G).$$

On the other hand, if m is negative then

$$\rho_{(g,\phi)}^{mn} = \left(\rho_{g,\phi}^{-mn}(1_G)\right)^{-1}(1_G) = \left(\rho_{(\rho_{(g,\phi)}^n(1_G),\phi^n)}^{-m}\right)^{-1}(1_G) = \rho_{(\rho_{(g,\phi)}^n(1_G),\phi^n)}^m(1_G).$$

□

Theorem 4 (Recursion tool, [25]). *Let G be a finite group, $\phi \in \text{Aut}(G)$ and $g, h \in G$. Given a ϕ -invariant normal subgroup N , set $\bar{\phi}$ to be the induced automorphism on G/N and $\phi|_N$ the induced automorphism on N . Then*

$$\text{SDLP}(G, \phi, g, h) = (t_0 + t_1 n_0) + (n_1 n_0)\mathbb{Z},$$

where

$$\text{SDLP}(G/N, \bar{\phi}, gN, hN) = t_0 + n_0\mathbb{Z}$$

and

$$\text{SDLP}(N, (\phi|_N)^{n_0}, \rho_{(g,\phi)}^{n_0}(1_G), (\rho_{(g,\phi)}^{t_0})^{-1}(h)) = t_1 + n_1\mathbb{Z}.$$

Proof. As N is ϕ -invariant it follows that $\phi|_N \in \text{Aut}(N)$ and $\bar{\phi}(gN) := \phi(g)N$ is a well defined automorphism of G/N . In the group G/N , for any $f \in G$, it follows that

$$(\rho_{(g,\phi)}(f))N = (g\phi(f))N = (gN)\phi(fN) = \rho_{(gN,\bar{\phi})}(fN)$$

and thus inductively it can be shown that $(\rho_{(g,\phi)}^x(f))N = \rho_{(gN,\bar{\phi})}^x(fN)$ for any integer x .

Assume $h = \rho_{(g,\phi)}^x(1_G)$ for some x . Then $hN = \rho_{(g,\phi)}^x(1_G)N = \rho_{(gN,\bar{\phi})}^x(1_{G/N})$. In other words $x \in \text{SDLP}(G/N, \bar{\phi}, gN, hN) = t_0 + n_0\mathbb{Z}$. Hence it suffices to compute the set of all t such that $h = \rho_{(g,\phi)}^{t_0+t n_0}(1_G)$.

By applying the second property from Lemma 1,

$$h = \rho_{(g,\phi)}^{t_0+t n_0}(1_G) \iff (\rho_{(g,\phi)}^{t_0})^{-1}(h) = \rho_{(g,\phi)}^{t n_0}(1_G) = \rho_{(\rho_{(g,\phi)}^{n_0}(1_G),\phi^{n_0})}^t(1_G).$$

Moreover, by Theorem 2.5, the definition of n_0 implies that $\rho_{(g,\phi)}^{n_0}(1_G)N = \rho_{(gN,\bar{\phi})}^{n_0}(1_{G/N}) = 1_{G/N}$. In other words $\rho_{(g,\phi)}^{n_0}(1_G) \in N$. Thus $h = \rho_{(g,\phi)}^{t_0+t n_0}(1_G)$ if and only if $t \in \text{SDLP}(N, (\phi|_N)^{n_0}, \rho_{(g,\phi)}^{n_0}(1_G), (\rho_{(g,\phi)}^{t_0})^{-1}(h))$. In particular,

$$\text{SDLP}(G, \phi, g, h) = t_0 + n_0(t_1 + n_1\mathbb{Z}) = t_0 + n_0 t_1 + n_0 n_1 \mathbb{Z}.$$

□

We can now consider applying this tool to reduce the general case of SDLP, via a composition series, to the case of SDLP in simple groups.

To determine $\text{SDLP}(G, \phi, g, h)$ via the application of Theorem 4, we need to construct the following: a ϕ -invariant normal subgroup N of G ; the quotient G/N ; the induced map $\bar{\phi}$ on the quotient; and the integer n_0 . We assume that given $N \triangleleft G$, constructing G/N can be done efficiently. Moreover, [25] describes a general method of evaluating the induced map $\bar{\phi}$. The computation of the integer n_0 can be done with a Shor-like algorithm by Theorem 3. Thus the main obstacle is the computation of the ϕ -invariant normal subgroup.

3.2 Computing Automorphism Invariant Normal Subgroups

The purpose of this section is to describe an algorithm that computes the invariant subgroups. The technique can be understood as building a machine taking as input some maximal normal subgroup of the group in which we wish to address SDLP, and outputting a ϕ -invariant subgroup of the maximal normal subgroup. The techniques for computing maximal normal subgroups in arbitrary finite groups are taken from the literature, which does not appear to be entirely resolved on this subject. For now, we assume we have an oracle $\Gamma()$, that on input of a black-box description of a group G , outputs a black-box description of one of its maximal normal subgroups. Discussion of the methods in the literature for implementing such an oracle are delayed to Appendix A.

Our method consists of showing that either we can compute a ϕ -invariant normal subgroup from an arbitrary maximal normal subgroup, or G has no characteristic subgroups (that is, it is “characteristically simple”) - and it is well known (see [48, Lemma 2.8]) that a group is characteristically simple if and only if it is isomorphic to S^k , where S is a simple group. In this latter case we have two sub-cases: either G is abelian, or ϕ acts transitively on the k factors of G , allowing a bespoke method of reduction.²

A method of computing ϕ -invariant normal subgroups from an arbitrary maximal normal subgroup N is given in [25], and works as follows. Set $N_1 = N$ and for $i \geq 2$ define $N_i = N_{i-1} \cap \phi^{i-1}(N)$. This sequence must eventually stabilize, say for some integer $j \in \mathbb{N}$: it is not difficult to show that N_j is ϕ -invariant, and that, since each intersection is a subgroup, we arrive at this stabilization within $\log |G|$ steps. For brevity we will refer to this method as the “intersection trick”.

Notice that we are not *a priori* guaranteed that the output of the intersection trick is non-trivial (certainly the trivial subgroup is ϕ -invariant). The intersection trick, however, will not terminate with the trivial subgroup if the maximal normal subgroup we started with contains a G -characteristic subgroup, since such a G -characteristic subgroup is also contained in the image of N under any automorphism, by definition. It would therefore suffice to demonstrate that a non-characteristically simple group is such that every maximal normal subgroup contains a characteristic subgroup in G . In fact, we are able to provide this alternate classification of the characteristically simple groups, as shown below.

Lemma 2. *Let G be a finite group. G possesses a non-trivial G -characteristic subgroup if and only if every maximal normal subgroup N of G contains a non-trivial G -characteristic subgroup.*

Proof. The reverse direction is trivial. Assume then that G is not characteristically simple and contains a maximal normal subgroup N . We show that N contains a nontrivial characteristic subgroup of G .

Consider the subgroup $\mathcal{J}(G)$ defined as the intersection of all maximal normal subgroups, known as the “Jacobson radical” of G . By definition, $\mathcal{J}(G)$ is

² The situation is actually slightly more complicated than this, as we will see.

contained in N and $\mathcal{J}(G)$ is characteristic. Hence it can be assumed that $\mathcal{J}(G)$ is trivial, which implies G is a direct product of simple groups by [5, Remark 4.8].

Set $G = S_1^{a_1} \times \dots \times S_n^{a_n}$, with $S_i \not\cong S_j$ for $i \neq j$. As G is not characteristically simple, $n \geq 2$. Assume S_1, \dots, S_m are non-abelian and S_{m+1}, \dots, S_n are abelian, so that the centre of G is given by $Z(G) = \prod_{i=m+1}^n S_i^{a_i}$. Additionally, write each factor as $S_i^{a_i} = S_{i,1} \times \dots \times S_{i,a_i}$.

If N is a maximal normal subgroup, then there exists some pair (i, j) such that $S_{i,j} \not\subseteq N$. By normality, $[N, S_{i,j}] \leq N \cap S_{i,j} = 1$. It follows that $G \cong N \times S_{i,j}$. Hence for any $k \neq i$ it follows that $S_k^{a_k} \subseteq N$, as otherwise there is some l such that $G \cong N \times S_{k,l}$ implying that $S_{k,l} \cong S_{i,j}$. To prove the statement, it thus suffices to show that for each $1 \leq k \leq n$ the subgroup $S_k^{a_k}$ is characteristic in G .

Assume first $k \leq m$ and $\phi \in \text{Aut}(G)$. Let $1 \leq j, j' \leq a_k$. If $\phi(S_{k,j}) \cap S_{k,j'} = 1$ then $[\phi(S_{k,j}), S_{k,j'}] = 1$. Thus if $\phi(S_{k,j}) \cap S_{k,j'} = 1$ for all j' , then $\phi(S_{k,j}) \leq C_G(S_k^{a_k}) = \prod_{j \neq k} S_j^{a_j}$; which yields a contradiction as $C_G(S_k^{a_k})$ has no composition factor isomorphic to S_k . Thus there must exist some j' such that $\phi(S_{k,j}) = S_{k,j'}$ and so $\phi(S_k^{a_k}) = S_k^{a_k}$.

Finally consider $k \geq m + 1$. As the S_k are non-isomorphic groups, each $S_k^{a_k}$ must be the unique Sylow p_k subgroup of $Z(G)$ for some prime p_k . Therefore $S_k^{a_k}$ is characteristic in G as being characteristic is transitive. \square

Notice that if the intersection trick terminates with the identity, by Lemma 2, G is characteristically simple. However, there are situations where a maximal normal subgroup of a characteristically simple group contains a ϕ -invariant normal subgroup. Whether or not this happens, in the non-abelian case, is related to the the automorphism ϕ . In particular, for S a non-abelian simple group, $\text{Aut}(S^k) \cong \text{Aut}(S)^k \wr \text{Sym}(\{1, \dots, k\})$; in other words, every automorphism in $\text{Aut}(S^k)$ can be thought of as possessing a unique permutation component.

Lemma 3. *Let G be a non-abelian finite group, and ϕ one of its automorphisms. The intersection trick for determining a ϕ -invariant normal subgroup from a maximal normal subgroup of G terminates in the trivial subgroup if and only if the group $G \cong S^k$ for some non-abelian simple group S with $k \in \mathbb{N}$, and the permutation component of ϕ is a k -cycle.*

Proof. First, we note that the normal subgroups of a non-abelian characteristically simple group S^k are exactly the subgroups $\prod_{j=1}^l S_{i_j}$, where $\{i_1, \dots, i_l\} \subset \{1, \dots, k\}$. In other words, every normal subgroup of S^k corresponds uniquely with a subset of $\{1, \dots, k\}$. Clearly, the maximal normal subgroups of S^k correspond to the subsets of $\{1, \dots, k\}$ of size $k - 1$.

Set $G = S^k$ for S a non-abelian simple group and suppose the permutation component of ϕ is a k -cycle. Since the maximal normal subgroup N we give as input to the intersection trick is of the form $\prod_{i=1, i \neq j}^k S_i$ for some $j \in \{1, \dots, k\}$, we have that

$$\{\phi^i(N) : i \in \mathbb{N}\} = \left\{ \prod_{i=1, i \neq j}^k S_i : 1 \leq j \leq k \right\}$$

The intersection of all these subgroups is trivial, and so we are done in this direction.

Now suppose that the intersection trick terminated in the trivial subgroup. We have already seen that the group G must, in this case, be characteristically simple, and so without loss of generality is of the form S^k , where S is a non-abelian simple group. Consider a maximal normal subgroup N of G . We are going to argue that if the permutation component of ϕ , which we will denote σ_ϕ , is not a k -cycle, then N will contain a non-trivial, ϕ -invariant normal subgroup, and so the intersection trick could not have had as output the trivial subgroup - a contradiction.

To see this, consider the orbits of the permutation σ_ϕ (that is, the distinct subsets of $\{1, \dots, k\}$ that are invariant under σ_ϕ). Of course, σ_ϕ is a k -cycle if and only if it has a single orbit; suppose that it has strictly more than one. Denote by \mathcal{I}_N the size $k - 1$ subset of $\{1, \dots, k\}$ corresponding to N under the bijection alluded to above. Because \mathcal{I}_N has size $k - 1$ and there are two or more orbits, \mathcal{I}_N must contain one of the orbits. Consider the normal subgroup corresponding to this orbit; since the orbit is fixed under the permutation σ_ϕ , the corresponding subgroup, say N' , is fixed under ϕ . Now, the ϕ -invariant normal subgroup N' is contained in N , so the intersection trick will terminate in a subgroup no smaller than N' . In particular, the intersection trick did not terminate in the trivial subgroup, giving the desired contradiction. \square

We are now ready to give the algorithm computing ϕ -invariant normal subgroups, given a maximal normal subgroup. In the case that no ϕ -invariant normal subgroup can be found, our algorithm outputs its input as a characteristically simple group, and determines whether this characteristically simple group is abelian or not.

Theorem 5. *Let G be a finite black-box group, and suppose ϕ is an automorphism of G . Given an oracle computing maximal normal subgroups, Algorithm 1 either computes a non-trivial ϕ -invariant subgroup of G , or detects that G is characteristically simple. If characteristic simplicity is detected, the algorithm also detects whether the group was abelian or not. In any case the algorithm finishes in time quantum polynomial in $\log |G|$.*

Proof. Let N be a maximal normal subgroup of G obtained from the oracle Γ . If N contains a non-trivial characteristic subgroup of G then, since this characteristic subgroup will also be contained in $\phi^i(N)$ for every $i \in \mathbb{N}$, the intersection trick will not terminate with the trivial subgroup.

If it does terminate with the trivial subgroup, we have already seen that the group we started with must be characteristically simple. If it is abelian, then, it is elementary abelian, and there are efficient quantum methods of recognising elementary abelian groups. If this test is failed we indicate instead that we have a non-abelian characteristically simple group. \square

Before moving on to the full reduction, we note that in the case that G is abelian and characteristically simple, the structure of the automorphisms is more

Algorithm 1. (Inv): Computing ϕ -invariant normal subgroups, or detecting either flavour of characteristically simple group.

Input: G, ϕ , oracle Γ computing maximal normal subgroups

Output: ϕ -invariant $N \triangleleft G$ or G

```

1:  $N \leftarrow \Gamma(G)$ 
2:  $N_1 \leftarrow N$ 
3:  $N_2 \leftarrow \phi(N)$ 
4:  $j \leftarrow 2$ 
5: while  $N_j \neq N_{j-1}$  do
6:    $j \leftarrow j + 1$ 
7:    $N_{j+1} \leftarrow N_j \cap \phi^{j-1}(N)$ 
8: end while
9: if  $N_j \neq \{1\}$  then
10:  return  $N_j$ 
11: else if  $G$  abelian then
12:   $G.\text{CS\_Abelian\_Flag} \leftarrow 1$  return  $G$ 
13: else
14:   $G.\text{CS\_NonAbelian\_Flag} \leftarrow 1$  return  $G$ 
15: end if

```

complicated than the structure described in Lemma 3, that is, $\text{Aut}((\mathbb{Z}/p\mathbb{Z})^n) \cong \text{GL}_n(\mathbb{F}_p)$. In order to avoid dealing with this algebraically, we can now simply outsource the abelian case to the method of [25] for solving SDLP in an elementary abelian group. Otherwise, the group and automorphism we started with have the form described in Lemma 3. We develop an algorithm for handling this case below.

Lemma 4. *Suppose G is a finite, non-abelian, characteristically simple group and ϕ is one of its automorphisms. We have that $G = S^k$ for S some non-abelian simple group and $k \in \mathbb{N}$; suppose moreover that the permutation component of ϕ is a k -cycle. Denote by $[g]_i$ the i -th coordinate of an element in the direct product group. Provided access to an oracle Θ for solving SDLP in simple groups, Algorithm 2 solves $\text{SDLP}(G, \phi)$ efficiently, with at most k^2 calls to the oracle.*

Proof. First note that by [1, Theorem 5.1], we can decompose G into its non-abelian simple factors. As such we can talk about projections of G onto its co-ordinates, and assume knowledge both of the integer k and black box representation of the simple factor S .

We know that the permutation component of ϕ is a k -cycle, so ϕ^k must consist only of co-ordinate-wise application of automorphisms in $\text{Aut}(S)$. Call these permutations to be applied co-ordinate wise $\phi^k = (\phi_1, \dots, \phi_k)$. Set also $(g_1, \dots, g_k) = \rho_{(g, \phi)}^k(1_G)$.

We wish to find the integers x such that $h = \rho_{(g, \phi)}^x(1_G)$. Of course, any such integer is of the form $i + kt$ for $0 \leq i < k$. Defining $(h_{i,1}, \dots, h_{i,j}) = \rho_{(g, \phi)}^{-i}(h)$, for

any $i \in \{0, \dots, k - 1\}$, have

$$\begin{aligned} h = \rho_{g,\phi}^x(1_G) &\iff \rho^{-i}(h) = \rho_{(g,\phi)}^{kt}(1_G) \\ &\iff (h_{i,1}, \dots, h_{i,k}) = \rho_{(\rho_{g,\phi}^k(1_G), \phi^k)}^t(1_S, \dots, 1_S) \\ &\iff (h_{i,1}, \dots, h_{i,k}) = (\rho_{(g_1, \phi_1)}^t(1_S), \dots, \rho_{(g_k, \phi_k)}^t(1_S)) \end{aligned}$$

In other words, given an $i \in \{0, \dots, k - 1\}$, we get k instances of SDLP in S that we can input to the SDLP oracle Θ . Any value t that solves all k of these instances is such that $x = i + kt$ has $h = \rho_{(g,\phi)}^x(1_G)$. In order to find the solutions of this latter instance of SDLP, then, it suffices to check the k problem instances defined by all k choices of i , giving k^2 total calls to the oracle. This procedure is outlined in Algorithm 2. \square

Algorithm 2. (*CSimple*): Solving particular instances of SDLP in non-abelian, characteristically simple groups.

Input: G, ϕ, g, h

Output: Element of solution set of $\text{SDLP}(G, \phi)$ for g, h

- 1: $S \leftarrow$ non-abelian simple factor of G
 - 2: $k \leftarrow$ number of copies of S
 - 3: $(\phi_1, \dots, \phi_k) \leftarrow \phi^k$
 - 4: Solutions $\leftarrow \{\}$
 - 5: **for** i **from** 0 **to** $k - 1$ **do**
 - 6: $h_{i,1} \leftarrow [\rho_{(g,\phi)}^{-i}(h)]_1$
 - 7: SubSolutions $\leftarrow \Theta(S, \phi_1, g_1, h_{i,1})$
 - 8: **for** j **from** 2 **to** k **do**
 - 9: $h_{i,j} \leftarrow [\rho_{(g,\phi)}^{-i}(h)]_j$
 - 10: SubSolutions \leftarrow SubSolutions $\cap \Theta(S, \phi_j, g_j, h_{i,j})$
 - 11: **end for**
 - 12: Solutions \leftarrow Solutions $\cup \{i + k \cdot \text{SubSolutions}\}$
 - 13: **end for**
-

3.3 The Decomposition Algorithm

We are now ready to provide our reduction to simple groups.

Theorem 6. Consider $\text{SDLP}(G, \phi)$ for some finite group G , one of its automorphisms ϕ , and group elements g, h . Suppose we have an oracle Γ computing maximal normal subgroups of G . Suppose, moreover, that we have an oracle Θ that, on input of the data S, ν, g, h for S a simple group, ν one of its automorphisms, and $g, h \in S$, outputs the set of solutions of $\text{SDLP}(S, \psi)$ for g, h . There exists an algorithm $\text{Solve}()$ that has the following properties: the algorithm terminates in time polynomial in $\log |G|$, having made logarithmically many calls to Θ ; and outputs a solution of $\text{SDLP}(G, \phi)$. The algorithm $\text{Solve}()$ is defined as in Algorithm 3, where $\phi, n_0, g', h', \bar{\phi}$ and ψ have the same meaning as in the proof of Theorem 4.

Algorithm 3. Solve(G, ϕ, g, h)

Input: (G, ϕ, g, h), oracles Γ, Θ

Output: (t, n) such that $SDLP(G, \phi, g, h) = t + n\mathbb{Z}$

```

1:  $N \leftarrow Inv(G, \phi)$  ▷ Algorithm 1
2: if  $N.CS\_Abelian\_Flag == 1$  then
3:   ( $t, n$ )  $\leftarrow$  solutions obtained from [25] method of solving SDLP in elementary
   abelian groups
4: else if  $N.CS\_NonAbelian\_Flag == 1$  then
5:   ( $t, n$ )  $\leftarrow$  CSimple( $G, \phi, g, h$ ) ▷ CSimple (Algorithm 2) can access  $\Theta$ 
6: else
7:   ( $t_0, n_0$ )  $\leftarrow$  Solve( $G/N, \bar{\phi}, \psi(g), \psi(h)$ )
8:   ( $t_1, n_1$ )  $\leftarrow$  Solve( $N, \phi^{n_0}, g', h'$ )
9:   ( $t, n$ )  $\leftarrow$  ( $t_0 + t_1 n_0, n_0 n_1$ )
10: end if
11: return ( $t, n$ )

```

Proof. We verify that the algorithm terminates after at most $\log|G| - 1$ internal repetitions of Solve(). Start with G : if it is not simple, there are two cases. If the group is characteristically simple, this is detected by the algorithm Inv defined in Algorithm 1 (which implicitly calls Γ), and there are two sub-cases. Either the $CS_Abelian_Flag$ attribute is set to 1 by Inv , and we can solve the problem instance by applying the method of [25] for solving SDLP in an elementary abelian group; or $CS_NonAbelian_Flag$ is set to 1, and we solve the problem instance with Algorithm 2. If characteristic simplicity is not detected, Algorithm 1 computes a ϕ -invariant subgroup N , and we run Solve() on the two induced problems defined in N and G/N . For these groups, if they are not simple, repeat the procedure, and so on.

As each normal subgroup of G/N is of the form M/N and $(G/N)/(M/N) \cong G/M$, it follows that the internal repetitions of Solve() reduces the problem to solving instances Solve($N_i/N_{i-1}, \phi_i, g_i, h_i$) for a subnormal series $1 \triangleleft N_1 \triangleleft \dots \triangleleft N_n = G$ such that N_i/N_{i-1} is either abelian or has no ϕ_i -invariant subgroup for suitable automorphisms ϕ_i and elements g_i and h_i . Moreover, as each N_i/N_{i-1} has order at least 2 it follows that $n \leq \log|G|$ and thus Solve() must terminate after at most $\log|G|$ internal repetitions. □

It now remains to develop methods for solving SDLP in simple groups. The rest of the paper will be devoted to this effort.

4 Reduction to Matrix Power Problem

In this section, we present a rather generic method of solving SDLP—indeed, it is defined for any group. We build on the ideas of [25, Theorem 8], which provides a reduction of SDLP in some finite group G , to the matrix power problem in the case that the group G is a matrix group over a field. Our observation is that,

by looking at the linear representations of an arbitrary group, there is a sense in which *every* group is a matrix group over a field. Moreover, in the case where ϕ is inner, we are able to compute a linear map that “mimics” the effect of $\rho_{(g,\phi)}$, thereby allowing us to apply the same techniques given by [25, Theorem 8]. It turns out that simple groups are well-suited to the application of this method, because the outer automorphism group of a simple group in general remains quite small.

Let us first outline the intuition behind the method: first, by Cayley’s theorem, we know that every finite group G admits a faithful linear representation³; that is, an injective group homomorphism $G \rightarrow \text{GL}_n(K)$ for some field K . Now, $\text{GL}_n(K)$ lives in the ambient space $M_n(K)$, the matrix algebra of all $n \times n$ matrices with entries in the field K . We can think of this space as an n^2 -dimensional vector space equipped with the natural addition and scalar multiplication, so we can imagine that we have a linear map T on this vector space. Suppose that this map T is such that $T \circ \psi = \psi \circ \rho_{(g,\phi)}$; we then immediately have that $T^i \circ \psi = \psi \circ \rho_{(g,\phi)}^i$. It follows that, in order to solve the SDLP instance, it suffices to find an integer x such that $T^x \cdot \psi(1_G) = \psi(h)$, where $\psi(1_G)$ is a vector in the n^2 -dimensional vector space, and \cdot refers to the usual notion of multiplication of a matrix by a vector. We have arrived at an instance of the so-called *matrix power problem*; when the matrices are invertible we have the same reduction to the period-finding routine of Shor’s algorithm as one has for the standard discrete logarithm problem, and so we have a solution in quantum polynomial time.

If instead we have a projective linear representation, i.e., an injective homomorphism $G \rightarrow \mathbb{PGL}_n(K)$ the same reduction can be applied to projective matrices in $\mathbb{PGL}_{n^2}(K)$.

Lemma 5. *Let G be a finite group, and $\psi : G \rightarrow \text{GL}_n(K)$ and $\bar{\psi} : G \rightarrow \mathbb{PGL}_n(K)$ a (projective) linear representation. Given an instance of SDLP for G and ϕ , where ϕ is an inner automorphism, i.e., $\phi(g) = m g m^{-1}$ for some $m \in G$, define the linear map $\mathbf{T} : M_n(K) \rightarrow M_n(K)$, $M \mapsto \psi(gm)M\psi(m^{-1})$. Then \mathbf{T} descends to a map $\bar{\mathbf{T}} : \mathbb{P}M_n(K) \rightarrow \mathbb{P}M_n(K)$ and*

$$\mathbf{T} \circ \psi = \psi \circ \rho_{(g,\phi)} \text{ and } \bar{\mathbf{T}} \circ \bar{\psi} = \bar{\psi} \circ \rho_{(g,\phi)} \tag{1}$$

Proof. Since \mathbf{T} is linear, it clearly descends to a map $\bar{\mathbf{T}}$ as described. Let $h \in G$, then by definition

$$(\mathbf{T} \circ \psi)(h) = \psi(gm)\psi(h)\psi(m^{-1}) = \psi(gmhm^{-1}) = \psi(g\phi(h)) = (\psi \circ \rho_{(g,\phi)})(h)$$

The projective case follows immediately. □

We delay the discussion of the case in which the automorphism ϕ is outer. Armed with \mathbf{T} , the reduction to the matrix power problem works as follows.

³ Note that the dimension of the representation implied by Cayley’s theorem is rather large. For the groups we are interested in we will have to work harder than this to find lower-dimensional linear representations.

Lemma 6. *Given a finite group G together with an efficiently computable injective (projective) linear representation $\psi : G \rightarrow (\mathbb{P})\text{GL}_n(K)$, if ϕ is an inner automorphism, then we can reduce any SDLP instance to an instance of the matrix power problem in time polynomial in n .*

Proof. First suppose ψ is a linear representation. Given $h \in G$, we want to find $x \in \mathbb{N}$ such that $\rho_{(g,\phi)}^x(1_G) = h$. By Lemma 5, if ψ is faithful, this is equivalent to finding $x \in \mathbb{N}$ such that $T^x(\mathbf{a}) = \mathbf{b}$ where $\mathbf{a} = 1_{n \times n}$ (the $n \times n$ identity matrix) and $\mathbf{b} = \psi(h)$.

Let $W := \text{span}_K(T^i(\mathbf{a}) \mid i \geq 0)$, which is a K -linear subspace of $M_n(K)$. We define the K -linear map

$$\mathbf{S} : W \rightarrow W, \mathbf{v} \mapsto \mathbf{T}^x \mathbf{v}$$

Note that even though we do not know x , we can compute \mathbf{S} on W in polynomial time since we know $S(\mathbf{a}) = \mathbf{b}$. Note that $(\mathbf{T}|_W)^x = \mathbf{S}$, and since both \mathbf{S} and $\mathbf{T}|_W$ are known we can find x by solving the matrix power problem in $\text{GL}_{n^2}(K)$ (noting that \mathbf{S} and $\mathbf{T}|_W$ can be regarded as elements in $\text{GL}_{n^2}(K)$ after a choice of basis).

In the case that ψ is an injective projective representation, the result follows similarly, reducing SDLP to the matrix power problem in $\mathbb{P}\text{GL}_{n^2}(K)$. \square

Recall also that we did not have a method of computing the crucial map \mathbf{T} , should the automorphism in question not be inner. However, by [25, Proposition 2], we do have the option of taking the smallest power of the automorphism that is inner, say y , and instead solving at most y instances of SDLP for G and ϕ^y . It turns out, due to a result of Kohl [33, Theorem 1] that for simple groups one can expect this power to be small.

Theorem 7 (Kohl). *If G is a non-abelian finite simple group, then*

$$|\text{Out}(G)| < \log_2 |G|.$$

Since $\text{Out}(G) \cong \text{Aut}(G)/\text{Inn}(G)$ it follows that for any outer automorphism ϕ of a non-abelian finite simple group G there exists an integer x such that $\phi^x \in \text{Inn}(G)$; and crucially that this x is no larger than $\log_2 |G|$. We conclude the following.

Corollary 2. *Let G be a non-abelian finite simple group, and suppose we have an efficiently computable non-trivial (projective) linear representation $\psi : G \rightarrow (\mathbb{P})\text{GL}_n(K)$. Then we can solve SDLP in G , for any $\phi \in \text{Aut}(G)$, on a quantum computer in probabilistic polynomial time in $\log |G|$.*

Remark 2. Note that we did not have to insist in the above that the linear representation was faithful. In fact, any non-trivial representation of a simple group is faithful, since if the map were not injective it would have non-trivial kernel and therefore imply a proper normal subgroup of a simple group.

5 SDLP in Simple Groups

Now that we have an efficient reduction of the general case of SDLP to SDLP in simple groups, and a method of solving SDLP in simple groups whose complexity is a function of the faithful dimension in simple groups, let us review the known results in this area.

The classification of finite simple groups [48] says any finite simple group is isomorphic to one of the following:

1. A **cyclic group** of prime order p ;
2. A group of even permutations of a finite set of cardinality $n \geq 5$, also called **alternating group** Alt_n ;
3. A **classical group** of Lie Type:

$$\begin{aligned}
 \textit{Linear:} & \quad A_{n-1}(q) \cong \mathbb{P}\text{SL}_n(q), n \geq 2, \text{ except } \mathbb{P}\text{SL}_2(2) \text{ and } \mathbb{P}\text{SL}_2(3); \\
 \textit{Unitary:} & \quad {}^2A_{n-1}(q)\mathbb{P}\text{SU}_n(q), n \geq 3, \text{ except } \mathbb{P}\text{SU}_3(2); \\
 \textit{Symplectic:} & \quad C_n(q) \cong \mathbb{P}\text{Sp}_{2n}(q), n \geq 2, \text{ except } \mathbb{P}\text{Sp}_4(2); \\
 \textit{Orthogonal:} & \quad B_n(q) \cong \mathbb{P}\Omega_{2n+1}(q), n \geq 3, q \text{ odd}; \\
 & \quad D_n(q) \cong \mathbb{P}\Omega_{2n}^+(q), n \geq 4; \\
 & \quad {}^2D_n(q) \cong \mathbb{P}\Omega_{2n}^-(q), n \geq 4
 \end{aligned}$$

where q is a power p^a of a prime p ;

4. An **exceptional group** of Lie type:

$$G_2(q), q \geq 3; F_4(q); E_6(q); {}^2E_6(q); {}^3D_4(q); E_7(q); E_8(q)$$

where q is a prime power, or

$${}^2B_2(2^{2n+1}), n \geq 1; {}^2G_2(3^{2n+1}), n \geq 1; {}^2F_4(2^{2n+1}), n \geq 1$$

or the Tits group ${}^2F_4(2)'$

5. One of 26 **sporadic simple groups**.

For cyclic groups, SDLP is known to be equivalent to classical DLP, so we need to focus on the other families of groups. Our main tool for the infinite families is to show the existence of a linear representation to use Corollary 2, while for the sporadic groups (and the Tits group) we have a separate discussion in Sect. 5.2.

5.1 Infinite Families

For alternating groups and groups of Lie type, we show that they have a known efficient linear representation. Thus, if we have them in their “natural representation” (the explicit representation used in their textbook definitions), by Corollary 2 there is a quantum polynomial-time algorithm to solve SDLP.

However, it is possible that, even if we know the isomorphism class of a simple group, an isomorphism to the natural representation of the simple group may still be unknown or hard to compute. A classical example of this is elliptic

curves of prime order, which are known to be cyclic groups but require difficult discrete logarithm computations to actually map points to modular integers in a homomorphic way.

This is known in the group theory literature as the **Constructive Recognition Problem** [1, Section 9.2]; hence, for each family, we will discuss how to go from a simple black-box group G to an efficient linear representation. By *efficient* we mean that the complexity is polynomial in the string length of the black-box group elements and in the logarithm of the target group cardinality.

Alternating Groups. Alternating groups are the group of even permutations of a finite set of cardinality n . Since these are permutations, they act on any n -dimensional vector space by permuting the entries, and thus can be represented in $GL_n(K)$. Additionally, thanks to [27, Theorem 1], there is a probabilistic algorithm in time $O(n \log^2(n)N)$ to compute an isomorphism from any black-box group to the permutation representation of Alt_n , where N is the string length of the black-box group and a maximal n is provided. As a consequence of Corollary 2, we have the following result.

Lemma 7. *If G is a simple black-box group isomorphic to any alternating group Alt_n , for some known maximal n , we can solve SDLP for G in probabilistic polynomial time in $n \log |G|$ on a quantum computer.*

Groups of Lie Type. Following [22, Section 2], if S is a finite simple group of Lie type, then there exists an algebraic group $\mathbf{H} \leq GL_n(\overline{\mathbb{F}})$ over an algebraically closed field $\overline{\mathbb{F}}$ and a Steinberg endomorphism σ of \mathbf{H} such that $S \cong C_{\mathbf{H}}(\sigma)/Z(C_{\mathbf{H}}(\sigma))$. Note that there are 8 small cases where this group is not simple, however the only new non-abelian simple group which arises from these cases is the Tit’s group ${}^2F_4(2)$ (see [22, Definition 2.2.8 and Theorem 2.2.10]), which will be considered alongside the sporadic simple groups. Given a family of simple groups of Lie type ${}^\epsilon\Gamma_m(q)$ for any suitable ϵ and prime power q , the dimension n of the underlying algebraic group $\mathbf{H} \leq GL_n(\overline{\mathbb{F}}_q)$ is determined by Γ_m :

$$\frac{\Gamma_m}{n} \mid \frac{A_m}{m+1} \mid \frac{B_m}{2m+1} \mid \frac{C_m}{2m} \mid \frac{D_m}{2m} \mid G_2 \mid F_4 \mid E_6 \mid E_7 \mid E_8$$

Thus they are naturally described as subgroups of $\mathbb{P}GL_n(\mathbb{F}_q)$ (or $GL_n(\mathbb{F}_q)$ if the centre is trivial). This means that we can solve SDLP for such groups using a quantum computer as a consequence of Corollary 2.

Sadly, in contrast to the case of alternating groups, there is no plain polynomial-time algorithm to solve the constructive recognition problem, even if extensive literature has been written on it.

A series of works of Brooksbank and Kantor have proven that for all the families of classical groups (linear [15], unitary [13], symplectic [12] and orthogonal [14]), summarized in [21], we can efficiently compute isomorphisms to the natural representations of the groups under the availability of:

1. So called *number theory oracles*, computing discrete logarithms and factoring in polynomial time;
2. An oracle that, for any input black-box group G isomorphic either to $\text{SL}(2, q)$ or $\mathbb{P}\text{SL}(2, q)$, produces in time polynomial in $\log(q)$ an effective isomorphism $\text{SL}(2, q) \rightarrow G$.

Similarly, in [29, 30] the authors show how to compute, in polynomial time, isomorphisms for groups of exceptional Lie type, with the exception of large Ree groups ${}^2F_4(2^{2n+1})$ and even characteristic Steinberg triality groups of type ${}^3D_4(2^e)$, assuming the availability of number theory oracles and $\text{SL}(2, q)$ oracles as for classical types.

Since, thanks to Shor’s algorithm [46], we know that quantum computers can implement efficient *number theory oracles*, we can combine the previous results in the following lemma.

Lemma 8. *On a quantum computer, if G is a simple black-box group isomorphic to any group of Lie Type of characteristic q and dimension n , with the exception of ${}^2F_4(2^{2n+1})$ and ${}^3D_4(2^e)$, we can reduce SDLP for G in probabilistic polynomial time in n and $\log(q)$ to the constructive recognition problem for the group $\text{SL}(2, q)$.*

Constructive Recognition of $\text{SL}(2, q)$ Given its relevance for the general formulation of the problem, several works have studied $\text{SL}(2, q)$. For instance, the authors in [19] show how to compute an efficient isomorphism when the black-box group is a subgroup of the general linear group $\text{GL}_d(q^i)$, given discrete logarithm oracles.

In [2, Lemma 2.10], the authors are able to generalize the result even further, for the much wider class of black-box groups of quotients of matrix groups by recognizable normal subgroups, showing that $\text{SL}(2, q)$ can be constructively recognized in polynomial time having access to number theory oracles.

For general black-box groups, the problem has been solved in [31] for even characteristic and in [9] for the case of small characteristic $p \equiv 1 \pmod{4}$. For a general field, the research is partially open: actually, in the preprint [10], the authors show how to compute an isomorphism in polynomial time between the black-box group and $\text{SL}_2(\mathbb{K})$, where \mathbb{K} is black-box field isomorphic to \mathbf{F}_q , this last isomorphism can be clearly computed via the solution of discrete logarithms over \mathbb{K} . Although these last results would suffice to solve the problem, we await further review of these results among the community before drawing this conclusion definitively.

5.2 Sporadic Groups

There are 26 finite simple groups that do not fall into one of the infinite families and the Tits Group ${}^2F_4(2)'$. By the definition of $\rho_{(g, \phi)}$, it suffices to find $x \leq \max_{g \in G}(\text{ord}(g)) \cdot \max_{\phi \in \text{Aut}(G)}(\text{ord}(\phi))$. The ATLAS of finite groups [20] provides a complete list of element orders for sporadic groups and their automorphism

group. In particular, it follows that $\max_{g \in G}(\text{ord}(g)), \max_{g \in G}(\text{ord}(g)) \leq 119 < 2^7$ for each of these 27 groups.

Lemma 9. *For any sporadic finite simple group G and automorphism $\phi \in \text{Aut}(G)$, there is a brute force algorithm to solve SDLP for G, ϕ with at most 2^{14} multiplications in the holomorph of G .*

Adapting Shanks’ Baby-Step Giant-Step Algorithm. Adjusting Shanks’ Baby-Step Giant-Step (BSGS) algorithm [45] to our setting is a reasonably simple task. Knowing a modest-size upper bound N for the possible values of x , this can be a practical way to find x . Algorithm 4 shows the SDLP variant of the BSGS algorithm, and it is easy to verify that the algorithm stores $O(\sqrt{N})$ elements in the holomorph $G \rtimes \text{Aut}(G)$ and recovers the secret exponent x in $O(\sqrt{N})$ operations in $G \rtimes \text{Aut}(G)$.

Algorithm 4. Baby-step giant-step algorithm in $G \rtimes \text{Aut}(G)$.

Input: $(g, \phi) \in G \rtimes \text{Aut}(G)$, $h = (g, \phi)^x$, $N \in \mathbb{N}$ with $x \leq N$;

Output: the solution of x of the input SDLP instance.

```

1:  $n \leftarrow \lceil \sqrt{N} \rceil$ 
2:  $(s, t) \leftarrow ((g, \phi)^n, (1, id))$ 
3:  $T \leftarrow [(0, t)]$  ▷ Initialize table
4: for  $(j \leftarrow 1; j \leq n; j++)$ 
5:    $t \leftarrow t \cdot s$  ▷ Giant step
6:   Store  $(t, j)$  in  $T$ .
7: end for
8:  $(y, i) \leftarrow (h, 0)$ .
9: while  $(y, -)$  is not in  $T$  do
10:   $(y, i) \leftarrow (y \cdot (g, \phi)^{-1}, i + 1)$  ▷ Baby step
11: end while
12: return  $jn - i$  where  $(y, j)$  is in  $T$ .
```

We illustrate the algorithm with SDLP over \mathbb{M} .

Example 1. We implemented our BSGS algorithm in approximately 30 lines of Python using the `mmgroup` Python library [44], which offers an efficient implementation of \mathbb{M} . In all of our experiments, the running time did not exceed 5 s on a 2022 Macbook Air with 16 GB of RAM.

5.3 Determining the Isomorphism Type of a Black Box Simple Group

Note that for the alternating group, the recognition algorithm requires as input a maximum n such that G could be isomorphic to Alt_n , while the recognition algorithms for groups of Lie type require the isomorphism type of the black box

group. Therefore an important step to apply Lemma 7 and Lemma 8 requires finding out which recognition algorithm needs to be implemented on a given black box simple group. It turns out that nearly all simple groups (and characteristically simple) are characterised by their order.

Theorem 8. [32, Theorem 6.1] *Let S and T be non-isomorphic finite simple groups. If $|S^a| = |T^b|$ for some natural numbers a and b , then $a = b$ and S and T either are $A_2(4)$ and $A_3(2)$ or are $B_n(q)$ and $C_n(q)$ for some $n \geq 3$ and some odd prime power q .*

Once a black box simple groups order $|S|$ is known, it can be tested to which simple group does it coincide and then run the corresponding recognition algorithm, while if there is a collision it is only between two groups and thus both corresponding recognition algorithms could be run. For sporadic simple groups (and the Tits group), this is a direct test against 27 fixed values and for alternating groups this requires finding n such that $|S| = n!$. For finite groups of Lie type their orders are of the form $\frac{q^N}{m} \prod_{i=1}^n (q^{d_i} - \epsilon_i)$ and thus it suffices to find the valid values for q , N , n , m , d_i and ϵ_i . In particular, determining the simple groups with order equal to that of a given black box group is polynomial in $\log|S|$.

6 Conclusion

We conclude by giving a comprehensive overview of our results, and discussing the consequences for SDLP. We have also summarized the flow of our argument visually in Fig. 1; one can take this diagram as a map of the paper.

Consider a finite, black-box group G . Then, in quantum polynomial time (in $\log|G|$), we can reduce any SDLP in G instance to at most $\log|G|$ instances of SDLP in a simple group by using Sect. 3. As a corollary of the Classification of Finite Simple Groups, once the isomorphism type is known we can efficiently study each possible instance separately, employing two main attack tools: for infinite families, the results from Sect. 4; and for sporadic groups, an adapted version of the *Baby-Step Giant-Step algorithm* (Algorithm 4).

We see that, if the groups are given in their natural representations we can find linear representations and apply Corollary 2 to produce a solution to SDLP in the corresponding simple group S in quantum polynomial time in $\log|S|$, so SDLP on simple groups is no harder than the problem of computing an efficient linear representation starting from a black-box group. Even if not conclusive, the extensive group theory literature on the solution of the constructive recognition problem in probabilistic quantum polynomial time is enough evidence to conclude that SDLP on finite groups is not a reliable candidate for the construction of quantum resistant primitives.

We highlight that, from Fig. 1, we could get also constructive quantum probabilistic polynomial-time algorithms for solving SDLP in a finite, black-box group G if we solve these last open questions:

1. Provide constructive recognition algorithms for large Ree groups ${}^2F_4(2^{2n+1})$ and even characteristic Steinberg triality groups of type ${}^3D_4(2^e)$;
2. Have a clean, peer-reviewed discussion of the Constructive Recognition problem for $SL(2, q)$ on quantum computers.
3. Resolve the gaps in the literature on the computation of maximal normal subgroups (discussed in the appendix).

We close with some high-level remarks. It is perhaps not too surprising, that an arbitrary instance of SDLP reduces to SDLP instances in finite simple groups. However, the fact that *all* of these finite simple groups admit efficient methods of solving SDLP relies on the property that simple groups have low dimension and very small outer-automorphism groups. Recalling that the method of decomposition into finite simple groups could only fail when no characteristic subgroups were present, it is also rather unfortunate that this scenario coincides with the group being a direct product of simple groups, from which a different method of reduction is possible. The insecurity of SDLP in finite groups, in other words,

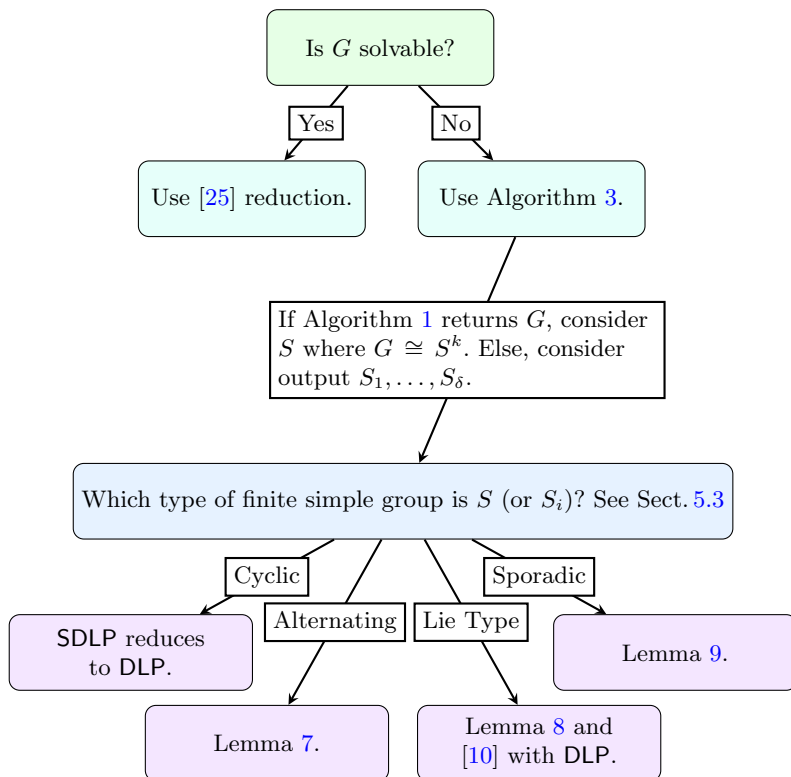


Fig. 1. Visual summary of a possible roadmap for a general SDLP instance over a finite group.

does not appear to result from some error in cryptographic design, but instead from fundamental properties of the finite groups themselves.

Acknowledgments. This collaboration was initiated during the “Post-Quantum Group-Based Cryptography” workshop at the American Institute of Mathematics (AIM), April 29-May 3, 2024. The authors are indebted to the workshop organizers Delaram Kahrobaei and Ludovic Perret and the AIM team for bringing this group together and creating a stimulating and collaborative atmosphere.

We want to thank Ray Perlner for spotting problems in the reasoning of an earlier version of this paper, and bringing those to our attention. We would also like to Gábor Ivanyos, with whom we had helpful correspondence. We also would like to acknowledge support by the following organizations: CB is supported by ONR Grant 62909-24-1-2002. GB is supported by SNSF Consolidator Grant CryptonIs 213766. DCST is partially supported by a grant from the Simons Foundation (712530, DCST). DJ is supported by an NSERC Alliance Consortia Quantum Grant (ALLRP 578463 – 22). LM is supported by an NSERC Canada Graduate Scholarship (Master’s). NH is supported by a gift from Google. RS is supported by NATO SPS project G5985. EP is supported by NCAE grant H98230-22-1-0328.

A Appendix: Finding Maximal Normal Subgroups

The task of finding a maximal normal subgroup depends on the particular implementation of the black-box group G . In general, if we know the particular structure of the group G , we may be able to recover them immediately from it. This can be done even with little knowledge, since from any subgroup S we can construct the smallest normal subgroup containing it via computing the normal closure $\langle S^G \rangle$ in linear time as explained in [3].

In the literature, several techniques are known to solve this task more systematically, via computing a composition series, in this way the first element in the series (starting from G) is our desired normal subgroup. However, this branch of literature typically wishes to achieve much stronger results, in particular without using quantum computers - we do not impose this limitation upon ourselves. To perform this calculation, aided by a quantum computer, we can:

- Use [25] if every non-Abelian composition factor of G possesses a faithful permutation representation of degree polynomial in the input size;
- Otherwise, [1, Theorem 1.1] gives us a quasi-composition series for G . Note that [1] requires a superset of the primes dividing the order of the group $|G|$ to solve the problem of computing order of group elements, with a quantum computer we can solve both these tasks. This result provides a quasi-composition chain $\{1\} \triangleleft G_{m-1} \triangleleft \dots \triangleleft G_1 \triangleleft G$, and tells us if G/G_1 is abelian, or simple and nonabelian. In the latter case, we have found a maximal normal subgroup $N = G_1$. In the former case, if $A = G/G_1$ has the unique encoding property, we can use [26, Theorem 6] on it, since abelian groups are solvable, i.e. $\nu(G) = 1$, and the procedure runs in quantum polynomial time. In this way we get the maximal normal subgroup $A_1 \triangleleft A$ from the composition

series, and A_1G_1 will be a maximal normal in G by the correspondence theorem. However, the general results from [1], does not immediately imply the unique-encoding property requested, so additional work may be required to solve this problem for the general case, even if in more concrete cases this may be practical.

In general, we do not expect that these problems should be of some fundamental computational difficulty. We leave the full resolution of the computation of maximal normal subgroups to further work.

References

1. László Babai and Robert Beals. A polynomial-time theory of black box groups i. *London Mathematical Society Lecture Note Series*, pages 30–64, 1999
2. László Babai, Robert Beals, and Ákos Seress. Polynomial-time theory of matrix groups. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 55-64, New York, NY, USA, 2009. Association for Computing Machinery
3. László Babai, Gene Cooperman, Larry Finkelstein, Eugene Luks, and Ákos Seress. Fast monte carlo algorithms for permutation groups. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 90–100, 1991
4. László Babai and Endre Szemerédi. On the complexity of matrix group problems i. In *25th Annual Symposium on Foundations of Computer Science, 1984.*, pages 229–240. IEEE, 1984
5. Reinhold Baer. Der reduzierte Rang einer Gruppe. *Journal für die reine und angewandte Mathematik*, 0214.0215: 146–173, 1964. URL <http://eudml.org/doc/150612>
6. Christopher Battarbee, Delaram Kahrobaei, Ludovic Perret, and Siamak F. Shahandashti. A subexponential quantum algorithm for the semidirect discrete logarithm problem, 2023
7. Christopher Battarbee, Delaram Kahrobaei, Ludovic Perret, and Siamak F. Shahandashti. Spdh-sign: Towards efficient, post-quantum group-based signatures. In Thomas Johansson and Daniel Smith-Tone, editors, *Post-Quantum Cryptography*, pages 113–138, Cham, 2023. Springer Nature Switzerland
8. Christopher Battarbee, Delaram Kahrobaei, and Siamak F Shahandashti. Semidirect product key exchange: The state of play. *Journal of Algebra and Its Applications*, page 2550066, 2023
9. Alexandre Borovik and Sukru Yalcinkaya. Steinberg presentations of black box classical groups in small characteristics, 2013
10. Alexandre Borovik and Şükrü Yalçınkaya. Natural representations of black box groups encrypting $sl_2(\mathbb{F}_q)$, 2020
11. Alexander Bors. A bound on element orders in the holomorph of a finite group, 2015
12. Peter A. Brooksbank. Fast constructive recognition of black box symplectic groups. *Journal of Algebra*, 320 (2): 885–909, 2008. ISSN 0021-8693. Computational Algebra
13. Brooksbank, Peter A.: Fast constructive recognition of black-box unitary groups. *LMS Journal of Computation and Mathematics* **6**, 162–197 (2003)
14. Brooksbank, Peter A., Kantor, William M.: Fast constructive recognition of black box orthogonal groups. *Journal of Algebra* **300**(1), 256–288 (2006)

15. Brooksbank, Peter A., Kantor, William M.: On constructive recognition of a black box $\text{psl}(d, q)$. *Groups and computation* **3**, 95–111 (1999)
16. Brown, Daniel, Koblitz, Neal, Legrow, Jason: Cryptanalysis of ‘make’. *J. Math. Cryptol.* **16**(1), 98–102 (2015)
17. Childs, Andrew M., Ivanyos, Gábor.: Quantum computation of discrete logarithms in semigroups. *J. Math. Cryptol.* **8**(4), 405–416 (2014)
18. Marston Conder and Charles R. Leedham-Green. Fast recognition of classical groups over large fields. *Groups and computation, III (Columbus, OH, 1999)*, 8: 113–121, 2001
19. Conder, Marston, Leedham-Green, Charles R., O’Brien, Eamonn: Constructive recognition of $PSL(2, q)$. *Trans. Amer. Math. Soc.* **358**(3), 1203–1221 (2006)
20. Conway, John H., Curtis, Robert T., Norton, Simon P., Parker, Richard A., Wilson, Robert A.: Atlas of finite groups. Oxford University Press, Eynsham (1985)
21. Dietrich, Heiko, Leedham-Green, Charles R., O’Brien, Eamonn A.: Effective black-box constructive recognition of classical groups. *Journal of Algebra* **421**, 460–492 (2015)
22. Daniel Gorenstein, Richard. Lyons, and Ron Solomon. *The classification of finite simple groups. Number 3. Part I.* American Mathematical Society, Providence, RI, 1998
23. Grigoriev, Dima, Shpilrain, Vladimir: Tropical cryptography ii: extensions by homomorphisms. *Communications in Algebra* **47**(10), 4224–4229 (2019)
24. Maggie Habeeb, Delaram Kahrobaei, Charalambos Koupparis, and Vladimir Shpilrain. Public key exchange using semidirect product of (semi)groups. In *International Conference on Applied Cryptography and Network Security*, pages 475–486. Springer, 2013
25. Muhammad Imran and Gábor Ivanyos. Efficient quantum algorithms for some instances of the semidirect discrete logarithm problem. *Designs, Codes and Cryptography*, 5 2024
26. Gábor Ivanyos, Frédéric Magniez, and Miklos Santha. Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 263–270, 2001
27. Sebastian Jambor, Martin Leuner, Alice C Niemeyer, and Wilhelm Plesken. Fast recognition of alternating groups of unknown degree. *Journal of Algebra*, 392: 315–335, 2013
28. Delaram Kahrobaei and Vladimir Shpilrain. Using semidirect product of (semi) groups in public key cryptography. In Arnold Beckmann, Laurent Bienvenu, and Nataša Jonoska, editors, *Pursuit of the Universal*, pages 132–141, Cham, 2016. Springer International Publishing
29. Kantor, W.M., Magaard, K.: Black box exceptional groups of Lie type. *Trans. Amer. Math. Soc.* **365**(9), 4895–4931 (2013)
30. Kantor, W.M., Magaard, K.: Black box exceptional groups of lie type ii. *Journal of Algebra* **421**, 524–540 (2015)
31. Kantor, William M., Kassabov, Martin: Black box groups isomorphic to $\text{pgl}(2, 2e)$. *Journal of Algebra* **421**, 16–26 (2015)
32. Kimmerle, Wolfgang, Lyons, Richard, Sandling, Robert, Teague, David N.: Composition factors from the group ring and artin’s theorem on orders of simple groups. *Proceedings of the London Mathematical Society* **3**(1), 89–122 (1990)
33. Stefan Kohl. A bound on the order of the outer automorphism group of a finite simple group of given order, 2003. Available at <https://stefan-kohl.github.io/preprints/outbound.pdf>

34. Leedham-Green, Charles R.: The computational matrix group project. *Groups and computation* **3**, 229–248 (2001)
35. Andrew Mendelsohn, Edmund Dable-Heath, and Cong Ling. A Small Serving of Mash: (Quantum) Algorithms for SPDH-Sign with Small Parameters. *Cryptology ePrint Archive*, Paper 2023/1963, 2023. URL <https://eprint.iacr.org/2023/1963>
36. Chris Monico. Remarks on MOBS and cryptosystems using semidirect products, 2021
37. Chris Monico and Ayan Mahalanobis. A remark on MAKE – a Matrix Action Key Exchange, 2020
38. Myasnikov, Alexei, Roman’kov, Vitalii: A linear decomposition attack. *Groups Complexity Cryptology* **7**(1), 81–94 (2015)
39. NIST. Post-Quantum Cryptography Standardization, 2017. URL: <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography>
40. Eamonn A O’Brien. Algorithms for matrix groups. *London Math. Soc. Lecture Note Ser.*, 388: 297–323, 2011
41. Rahman, Nael, Shpilrain, Vladimir: Make: A matrix action key exchange. *J. Math. Cryptol.* **16**(1), 64–72 (2022)
42. Nael Rahman and Vladimir Shpilrain. MOBS (Matrices Over Bit Strings) public key exchange. *Cryptology ePrint Archive*, Paper 2021 /560, 2021. URL <https://eprint.iacr.org/2021/560>
43. Vitalii Roman’kov. Linear decomposition attack on public key exchange protocols using semidirect products of (semi) groups, 2015
44. Martin Seysen. Python implementation of the monster group. GitHub repository, 2024. URL <https://github.com/Martin-Seysen/mmgrou>
45. Daniel Shanks. Class number, a theory of factorization, and genera. In *Proceedings of Symposia in Pure Mathematics*, 1971
46. Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994
47. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT ’97*, pages 256–266, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg
48. Robert A. Wilson. *The Finite Simple Groups*, volume 251 of *Graduate Texts in Mathematics*. Springer, 2009



Quantum Circuits of AES with a Low-Depth Linear Layer and a New Structure

Haotian Shi^{1,2} and Xiutao Feng¹(✉)

¹ Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China
{shihaotian,fengxt}@amss.ac.cn

² University of Chinese Academy of Sciences, Beijing, China

Abstract. In recent years quantum computing has developed rapidly. The security threat posed by quantum computing to cryptography makes it necessary to better evaluate the resource cost of attacking algorithms, some of which require quantum implementations of the attacked cryptographic building blocks. In this paper we manage to optimize quantum circuits of AES in several aspects. Firstly, based on de Brugière *et al.*'s greedy algorithm, we propose an improved depth-oriented algorithm for synthesizing low-depth CNOT circuits with no ancilla qubits. Our algorithm finds a CNOT circuit of AES MixColumns with depth 10, which breaks a recent record of depth 16. In addition, our algorithm gives low-depth CNOT circuits for many MDS matrices and matrices used in block ciphers studied in related work. Secondly, we present a new structure named compressed pipeline structure to synthesize quantum circuits of AES, which can be used for constructing quantum oracles employed in quantum attacks based on Grover's and Simon's algorithms. When the number of ancilla qubits required by the round function and its inverse is not very large, our structure will have a better trade-off of D - W cost. Moreover, our encryption oracle will have the lowest depth to date. We then give detailed encryption circuits of AES-128 under the guidance of our structure and make some comparisons with other circuits. Finally, the encryption part and the key schedule part have their own application scenarios. The Encryption oracle used in Simon's algorithm built with the former will have smaller round depth. For example, we can construct an AES-128 Encryption oracle with T -depth 33, while the previous best result is 60. A small variant of the latter, along with our method to make an Sbox input-invariant, can avoid the allocation of extra ancilla qubits for storing key words in the shallowed pipeline structure. Based on this, we achieve an encryption circuit of AES-128 with the lowest $TofD$ - W cost 130720 to date.

Keywords: Quantum circuit · Depth · AES · Encryption oracle

1 Introduction

Quantum computers provide a great potential of solving certain important information processing tasks that are intractable for any classical computer. Shor's algorithm [53] showed that a sufficiently large quantum computer allows to factor numbers and compute discrete logarithms in polynomial time, which represents an exponential speed-up compared to classical algorithms and can be devastating to many public-key encryption schemes in use today.

The possible emergence of large-scale quantum computing devices in the near future has brought new security threats and raised concerns about post-quantum security. Not only the public-key cryptosystem, the security of the symmetric-key cryptosystem is also under threat. A trivial application of Grover's algorithm [21] results in a quadratic speedup of the exhaustive search attack. Simon's algorithm [55] answers the question of how to find the period of a periodic function with n input bits in $O(n)$ quantum queries. As a result, many encryption structures and the most widely used modes of operation for authentication and authenticated encryption were attacked by using Simon's algorithm [29, 31]. Both of these two algorithms require the quantum oracle of the symmetric building block to be attacked. Moreover, the National Institute of Standards and Technology (NIST) used the complexity of the quantum circuit for AES with a bound of depth called MAXDEPTH as a baseline to categorize the post-quantum public-key schemes into different security levels in the call for proposals to the standardization of post-quantum cryptography¹. Both these reasons give rise to the growing appeals for studying the quantum implementations of symmetric-key building blocks as well as how to optimize them. This helps understand the quantum security of current encryption schemes and guides future post-quantum encryption designs.

The synthesis and optimization of quantum circuits have been studied for many years [4, 5, 27, 44, 50, 57]. Given an n -qubit unitary operator and an available gate set \mathcal{G} , synthesis algorithms find one of its implementations described as a sequence of G quantum gates in \mathcal{G} with width (number of qubits) W , full depth FD and T -depth TD . The optimization of G and W is related to the saving of resources and qubits, while the optimization of FD, TD is also concerned due to the phenomenon of quantum decoherence. In addition, it is worth noting that there is a lot of work on optimizing quantum circuits on some noisy intermediate-scale quantum (NISQ) devices (see [42, 58, 60] for an incomplete list). In the process of quantum computation, since it has been difficult to isolate qubits for a long time, qubits would interact unintentionally with external elements, which would distort the results. Assuming that two non-overlapping gates can run in parallel, the running time of the circuit is related to its depth. Therefore, the proper execution of complex algorithms can be significantly facilitated by optimizing the depth of quantum circuits since the decoherence time is very limited. The reduction of T -depth is more important in fault-tolerant computations where the running time is dominated by T -depth [19].

¹ <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf>.

Recent research on quantum implementation of symmetric ciphers mainly focuses on AES due to its popularity and importance. The main concerns are the structure, AES MixColumns and AES S-box. At the same time, there is also related work focusing on quantum implementations of other symmetric building blocks [62]. One research line is to optimize the width. Grassl *et al.* proposed the first quantum circuit of AES under the so-called zig-zag structure with low width [20]. Zou *et al.* proposed the improved zig-zag structure [63], and then Huang *et al.* presented the OP-based round-in-place structure with a similar idea [22]. Jaques *et al.* first proposed the straight-line structure of key schedule process with no ancilla states and first adopted the pipeline structure [24]. Furthermore, Jang *et al.* proposed the shallowed pipeline structure to reduce the full depth [23]. Li *et al.* directly designed an in-place quantum circuit of AES S-box with low width and constructed a straight-line circuit of AES with the lowest width to date [35]. During the research on quantum circuits of AES, many researchers studied low-width quantum circuits of AES S-box.

The other research line is to reduce the circuit depth. In Clifford+ T circuits, T -depth is the main concern since Clifford gates are much cheaper than the T gate. The pipeline structure, first mentioned in reversible logic implementations of AES in [17], is straightforward to provide a low T -depth circuit of AES in many studies. To synthesize a low T -depth AES S-box, usually a low Toffoli-depth Sbox which produces some redundant states is designed. Jaques *et al.* constructed an Sbox with Toffoli-depth 6, and then gave a quantum circuit of AES-128 with T -depth 120 [24]. Li *et al.* [34] proposed an Sbox with Toffoli-depth 4. Huang *et al.* also gave an Sbox with Toffoli-depth 4, and further gave one with Toffoli-depth 3, which is the theoretical minimum. Therefore, the T -depth of quantum circuit of AES-128 is reduced to 60 [22]. The width of Sbox was further reduced by Jang *et al.*'s and Liu *et al.*'s techniques [23,38], while the saving of qubits made the optimized Sbox no longer input-invariant.

Full depth is a forward-looking time-cost measure for quantum circuits, so optimizing the depth of a CNOT circuit can reduce the full depth of the entire circuit. CNOT circuits which consist only of CNOT gates appear as subcircuits of larger circuits, such as quantum oracles of symmetric ciphers, stabilizer circuits [1], and CNOT+ T circuits. Patel *et al.* and Jiang *et al.* proposed methods to generate CNOT circuits with asymptotic optimal gate count and space-depth trade-off, respectively [27,41]. For specific matrices, Xiang *et al.*'s method [59] is based on some reduction rules for matrix decomposition, can effectively reduce the number of gates of given CNOT circuits and provides CNOT circuits with the smallest CNOT gates to date for many MDS matrices and matrices used in block ciphers. A lot of work on quantum circuits of AES (see [22,23,35,63] for an incomplete list) adopted the implementation of AES MixColumns with 92 CNOT gates provided by Xiang *et al.*'s method. Its depth estimation given by the Q# resource estimator is 30. However, their method does not take the circuit depth into account. Zhu *et al.* defined the exchange-equivalence of sequences, and proposed a framework of optimizing the depth of a given CNOT circuit [61] by exploring the possibility of exchanging CNOT gates. They started the

optimization with the results of Xiang *et al.*'s method and gave a better estimation of depth 28 for AES MixColumns. Recently Liu *et al.* proposed a method for computing the depth of given quantum circuits and provided a circuit of AES MixColumns with depth 16 by computing the depth of many search results of Xiang *et al.*'s method. Some CNOT circuits of AES MixColumns with ancilla qubits are synthesized on the basis of optimized low-depth classical circuits, and the state-of-art classical circuit of AES MixColumns with minimum depth 3 requires 99 XOR gates [51]. In addition, de Brugière *et al.* proposed a depth-oriented greedy method and a block algorithm for small and middle scale matrices, respectively [18]. However, their methods have not been tested on the linear layers of many cryptographic building blocks.

1.1 Our Contributions

This paper mainly focuses on optimizing quantum circuits of AES and gives improvements in several aspects.

Improved Greedy Algorithm for Finding Low-Depth CNOT Circuits with No Ancilla Qubits. We first notice that related works of providing CNOT circuits of AES Mixcolumns either adopted non-depth-oriented search methods or determined the depth based on existing circuits. Instead, we use a depth-oriented search method. Since de Brugière *et al.* proposed a depth-oriented cost-minimization greedy algorithm that is suitable for random small scale matrices, we first apply their algorithm to AES Mixcolumns and find a circuit with depth 12, which is much better than a recent record of depth 16 in [38]. We then propose an improved greedy algorithm based on de Brugière *et al.*'s algorithm and successfully find a circuit with depth 10, which can be used to reduce the full depth of quantum circuits of AES. The improvement of our algorithm is reflected in three aspects. First, in addition to considering the logarithm of each row's Hamming weight, we also consider the *square* of each row's Hamming weight, which gives priority to rows or columns that are "far from being done" and is beneficial to reduce the circuit depth in many cases. Second, we treat two cases of row and column operations differently when evaluating the cost, that is, each column's Hamming weight is considered when column operations are performed. Finally, we give an equivalent condition of determining whether a matrix can be implemented with depth 1 to better handle sparse matrices. As applications, our improved greedy method provides low-depth CNOT circuits for many MDS matrices and matrices used in block ciphers (see Table 3, 4). Except for some matrices with depth 3, all the results are much better than those in [61]. De Brugière *et al.*'s algorithm is also applied to these matrices for comparison.

Compressed Pipeline Structure for Quantum Circuits of AES. We observe that the pipeline structure has a low depth but too many intermediate states, while the OP-based round-in-place structure has fewer intermediate states but a greater depth. To combine the advantages of the above two structures, we propose a new structure named compressed pipeline structure, which computes new states and eliminates intermediate states in parallel for qubit

reuse. If the round function is taken as a unit, our structure will have lower D - W cost than the above two structures when the number of ancilla qubits of a round function is small enough. To give detailed quantum circuits of AES-128, we propose iterative round functions for the encryption circuit under the guidance of our structure. Since two consecutive roundkeys are needed in the round functions, we also present a new circuit for the key schedule of AES-128 which provides linear components of consecutive two roundkeys in one round. Both cases of NCT-based circuit and qAND-based circuit are considered. Our circuit only needs such quantum circuits of AES S-box where the output register can only be set to $|0\rangle$ initially and has lower TD - W or $TofD$ - W cost when the number of ancilla qubits of AES S-box is small enough. The cost for the AES Grover oracle can be evaluated by referring to the cost of the encryption circuit.

The AES-128 Encryption Oracle with Lower T -Depth. The encryption circuit in our structure can be used to construct the Encryption oracle employed in Simon's algorithm with simplified cleaning of redundant states. If the round function is taken as a unit, our constructed Encryption oracle will have depth $r + 1$, which is almost half of the previous best result $2r$. When it comes to AES-128, the AES-128 Encryption oracle can be constructed with smaller T -depth. Since the redundant states of the encryption circuit can be cleaned by $|c\rangle$ with one layer of AES S-box, the AES-128 Encryption oracle can be constructed with T -depth 33, which breaks the previous record of T -depth 60 in [22].

Key Schedule of the Shallowed Pipeline Structure with Input-Invariant Sbox. In the key schedule of the shallowed pipeline structure, 10×32 qubits need to be allocated for storing the input register of low Toffoli-depth Sbox which cannot keep the input register unchanged. We find that adding some CNOT gates can make such Sbox input-invariant without increasing the Toffoli-depth and ancilla qubits, which ensures that the information of the input register is not lost. Based on this, we propose a new key schedule for the shallowed pipeline structure which is actually a small variant of the key schedule in the compressed pipeline structure. It can avoid the allocation of extra 10×32 qubits for storing key words and can be used to synthesize a quantum circuit of AES-128 with the lowest $TofD$ - W cost 130720 to date.

All the source codes and results of this paper are available at <https://gitee.com/Haotian-Shi/Quantum-circuits-of-aes-with-a-low-depth-linear-layer-and-a-new-structure>.

1.2 Organization

In Sect. 2 we introduce some background knowledge about quantum computation. In Sect. 3 we introduce some existing methods for optimizing the depth of CNOT circuits. Our new method and its application on some matrices are illustrated in Sect. 4. In Sect. 5 we propose our compressed pipeline structure for iterative building blocks and show its application in Encryption oracles and Grover oracles. Specific quantum implementations of AES in different scenarios and the resource costs are given in Sect. 6. We conclude our work in Sect. 7.

2 Preliminaries

2.1 Notations

We assume that the reader is familiar with AES [16]. Some notations used throughout the paper are listed as follows:

Table 1. Some notations used throughout the paper.

Notation	Definition
\mathbb{F}_2	The finite field with two elements 0 and 1
\oplus	The XOR computation
$GL(2, n)$	The set of all $n \times n$ invertible matrices over \mathbb{F}_2
$GF(n, \mathbb{F}_2)$	The finite field with 2^n elements
I_n	The n -by- n identity matrix over \mathbb{F}_2
$E(i + j)$	The resulting matrix by adding the j -th row to the i -th row of I_n (type-3 elementary matrix in $GL(2, n)$)
$E(i, j)$	The CNOT gate of adding the i -th qubit to the j -th qubit
$E(i \leftrightarrow j)$	The resulting matrix by exchanging the i -th and j -th row of I_n (type-1 elementary matrix in $GL(2, n)$), or the swapping of the i -th and j -th qubits
$ u\rangle$	A state vector u
$S(x)$	The function of AES S-box
$\mathcal{O}_{\mathcal{R}}$	The quantum oracle: $ x\rangle y\rangle \mapsto x\rangle y \oplus \mathcal{R}(x)\rangle$
$\mathcal{O}_{\mathcal{R}^{-1}}$	The quantum oracle: $ x\rangle y\rangle \mapsto x\rangle y \oplus \mathcal{R}^{-1}(x)\rangle$
k_j^l	The l -th (start from zero) 32-bit word of the j -th roundkey, or W_{4j+l} in the key schedule of AES

2.2 Quantum Computation

The simplest quantum system is a single qubit state. It can be described as a unit vector $|u\rangle$ in a Hilbert Space $\mathcal{H} = \mathbb{C}^2$ and has two computational basis states $|0\rangle$ and $|1\rangle$. Then $|u\rangle = \alpha |0\rangle + \beta |1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. An n -qubit state $|u\rangle$ can be described as a unit vector in $\mathcal{H}^{\otimes n}$, and a computational basis state can be described as a state of n -bit 0/1 string: $|x_1 x_2 \dots x_n\rangle$.

Several typical quantum gates include X, S, T, CNOT and Toffoli gates. In this paper we mainly focus on quantum circuits which compute classical vectorial boolean functions, so the input states we are concerned with are only computational basis states. The X gate, CNOT gate and Toffoli gate all act on computational basis states as shown in Fig. 1:

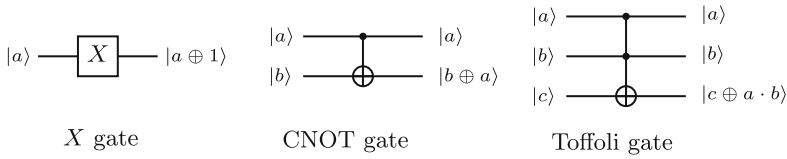


Fig. 1. Circuits of X gate, CNOT gate and Toffoli gate. The changed qubit is called the target qubit.

One can see that the roles they play in quantum computation are NOT gate, XOR gate and AND gate in classical computation, respectively.

There is another quantum gate called a quantum AND gate (qAND in short) which simulates the functionality of a classical AND gate. It differs from the Toffoli gate in that the target qubit must be $|0\rangle$. This gate together with its adjoint is illustrated in Fig. 2.

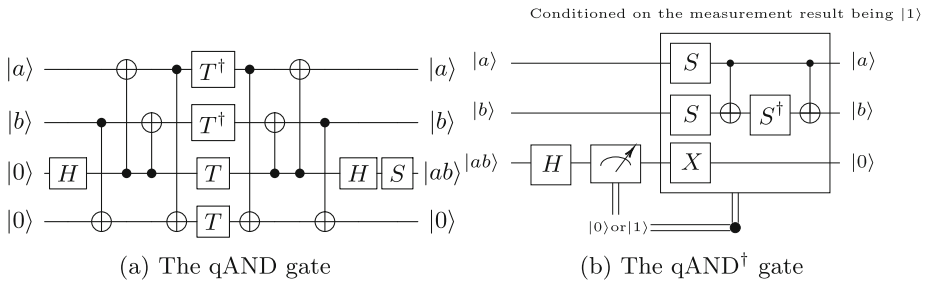


Fig. 2. The quantum AND gate together with its adjoint.

2.3 Optimization Goals

Due to the limited decoherence time and qubit resources, it is crucial to reduce the time cost and storage cost in quantum circuits. In circuits containing X , CNOT and Toffoli gates, the metrics of width (W), Toffoli depth ($TofD$), and $TofD \cdot W$ cost are crucial for evaluating the cost. In practice, the Toffoli gate can be decomposed with T -depth $3/4$ and full depth $9/8$, respectively, using 0 ancilla qubit [5], or decomposed with T -depth 1 using 4 ancilla qubits [49]. If the target qubit of a Toffoli gate is identically equal to $|0\rangle$, the Toffoli gate can be replaced by qAND with T -depth 1 using 1 ancilla qubit and its adjoint can be replaced by qAND[†] with T -depth 0 using 0 ancilla qubit. As is shown in [40], the Clifford gates are much cheaper than the T gate. Therefore, T -depth (TD) is a key parameter to measure the running time of a circuit. At the same time, a forward-looking perspective assumes that each gate has a unit depth and defines the full depth (FD) as a time-cost metric.

This paper mainly deals with quantum circuits of AES-128, including encryption circuits and Encryption oracles. Specifically, the encryption circuit is defined as:

$$|x\rangle |k\rangle |0\rangle \mapsto |x\rangle |k'\rangle |Enc_k(x)\rangle,$$

where $Enc_k(x)$ is the encryption of message x under the seed key k . In many cases the k' in $|k'\rangle$ is the roundkey of the last round. Note that $|x\rangle, |k\rangle$ can be in a superposition state. The encryption oracle differs from it in that the key register does not exist since the seed key is pre-fixed. It is defined as:

$$|x\rangle |0\rangle \mapsto |x\rangle |Enc(x)\rangle,$$

where $Enc(x)$ is the encryption of message x . In this paper, “quantum circuit” is seen as a general term for both two cases above.

3 State-of-Art Heuristics for Optimizing the Depth of CNOT Circuits

In this section, we first introduce some background knowledge of CNOT circuits. We then introduce some existing methods for optimizing the depth of CNOT circuits, including methods for handling existing circuits [38,61] and the depth-oriented greedy algorithm [18].

3.1 CNOT Circuits

A CNOT circuit is a quantum circuit that contains only CNOT gates. One can see from Fig. 1 that a CNOT gate adds one boolean tuple to another, so for an n -qubit system, the CNOT gate $E(i, j)$ controlled by the i -th qubit and targeting on the j -th qubit can be seen as the type-3 elementary matrix $E(j+i)$. Actually the linear layer of a cipher is an n -bit reversible linear boolean function and can be interpreted as an invertible matrix A in $GL(2, n)$. Therefore, the CNOT circuit of a linear layer A can be synthesized by referring to a proper form of matrix decomposition of A . Recall that the general form of matrix decomposition is illustrated below as Theorem 1:

Theorem 1. *Any A in $GL(2, n)$ can be expressed as a product of type-1 and type-3 elementary matrices.*

Note that the only type-2 matrix in $GL(2, n)$ is the identity matrix and therefore does not appear in the matrix decomposition. It is easy to see that the type-3 elementary matrices can be adjacent by the following swapping Property 1, as shown in Theorem 2.

Property 1. $E(i+j)E(k \leftrightarrow l) = E(k \leftrightarrow l)E(f_{k,l}(i) + f_{k,l}(j)), E(k \leftrightarrow l)E(i+j) = E(f_{k,l}(i) + f_{k,l}(j))E(k \leftrightarrow l)$, where

$$f_{k,l}(x) = \begin{cases} k, & \text{if } x = l; \\ l, & \text{if } x = k; \\ x, & \text{else.} \end{cases} \tag{1}$$

Theorem 2. Any A in $GL(2, n)$ can be expressed as

$$A = PE(i_1 + j_1)E(i_2 + j_2) \dots E(i_L + j_L), \quad (2)$$

where P is a permutation matrix.

Since the swapping of two qubits can be realized by rewiring for free, it is easy to convert a decomposition of A in Theorem 2 to a CNOT circuit of A and vice versa. Based on this, Xiang *et al.* defined the sequential XOR (s-XOR) metric which describes the minimum gate cost of implementing A by updating input variables to output variables:

Definition 1 (s-XOR). [59] Let $M \in GL(n, \mathbb{F}_2)$ be an invertible matrix. Assume (x_1, x_2, \dots, x_n) are the n input bits of M . It is always possible to perform a sequence of XOR instructions $x_i = x_i \oplus x_j$ with $1 \leq i, j \leq n$, such that the n input bits are updated to the n output bits. The s-XOR count of M is defined as the minimum number of XOR instructions to update the inputs to the outputs.

Xiang *et al.* proposed some reduction rules to reduce the number of type-3 elementary matrices in a matrix decomposition and obtained in-place implementations of the many constructed MDS matrices and matrices used in block ciphers with minimum XOR gates up to date [59]. The specific results of their method can serve as a fairly good starting point for optimizing the quantum depth of the corresponding CNOT circuits.

3.2 Computing the Depth of Given Circuits

Many efforts have been done on optimizing the depth of given CNOT circuits. Zhu *et al.* proposed an algorithm named One-way-opt, which involves iteratively extracts a layer of CNOT gates by exploring gates that can be exchanged forward and can run in parallel with the gates of the current layer [61]. One-way-opt is executed twice in both forward and backward directions of sequences. They compared their results with those given by the Q# resource estimator, which only explores the moving of parallelable gates. Recently, Jang *et al.* also adopted the idea of reordering gates to optimize the depth of linear layers [23]. Liu *et al.* proposed an algorithm FINDDEPTH to quickly determine the full depth of a given CNOT circuit [38]. It works by recording the updated qubits (target qubits) and used qubits (control qubits) of previous CNOT gates in each depth layer to determine the minimum depth of the current gate.

3.3 Greedy Method

De Brugière *et al.* proposed a depth-oriented greedy method to find low-depth CNOT circuits of invertible linear layers [18]. It is a cost-minimization algorithm, that is, a cost function needs to be defined to evaluate the cost of reducing the

matrix A to a permutation matrix, and then a strategy for exploring effective elementary transformations is designed according to the cost function.

The definition of the cost function is the only concern in the optimization of CNOT gate counts, while things are a little bit different when it comes to the optimization of the depth. Suppose a cost function to minimize has been defined to guide the search, the depth-oriented algorithm will find a series of elementary transformations of row layers and column layers to transform the target matrix A into a permutation matrix, as follows:

$$E_{i_d}^d E_{i_{d-1}}^d \cdots E_1^d \cdots E_{i_1}^1 E_{i_{1-1}}^1 \cdots E_1^1 A F_1^1 F_2^1 \cdots F_{j_1}^1 \cdots F_1^d F_2^d \cdots F_{j_d}^d = P,$$

where $E_1^t, E_2^t, \dots, E_{i_t}^t$ and $F_1^t, F_2^t, \dots, F_{j_t}^t$ for $t = 1, 2, \dots, d$ are layers of elementary row (column) operations that can act in parallel when converted to quantum CNOT gates, and P is a permutation matrix. Then one decomposition form in Theorem 2 of A is deduced as follows:

$$A = \tilde{P} \tilde{E}_1^1 \cdots \tilde{E}_{i_1-1}^1 \tilde{E}_{i_1}^1 \cdots \tilde{E}_1^d \cdots \tilde{E}_{i_d-1}^d \tilde{E}_{i_d}^d F_{j_d}^d F_{j_d-1}^d \cdots F_1^d \cdots F_{j_1}^1 F_{j_1-1}^1 \cdots F_1^1,$$

where \tilde{P}, \tilde{E}_q^p are obtained by Property 1. This decomposition corresponds to a CNOT circuit of A with depth $2d$ or $2d - 1$ ². One can see that unlike the case of the gate optimization, the depth-oriented algorithm searches row and column operations which reduce the cost function to generate a row layer and a column layer (sometimes only one row or column layer). Specifically, the authors defined two sets L_r and L_c , which record the recently applied row and column operations that can act in parallel. In each iteration of choosing an operation, only *available* row or column operations can be chosen, which are defined as follows:

Definition 2 (Available operation). *The available row (column, respectively) operations meet the following two conditions:*

- Reduce the cost function.
- Can act in parallel with recent operations in L_r (L_c , respectively).

If no available row or column operations exist, one resets L_r, L_c to empty. Each time a non-empty L_r or L_c is reset to empty, the depth count is increased by one. The algorithm ends when the cost function is equal to its minimum, that is, the current matrix is a permutation matrix, or when the depth counter exceeds a certain threshold, that is, the algorithm falls into a local minima.

Then for the definition of the cost function, the authors considered four choices to guide the optimization of the gate count in [13] and the depth in [18]:

- (1) $h_{sum}(A) = \sum_{i,j} a_{ij}$;
- (2) $H_{sum}(A) = h_{sum}(A) + h_{sum}(A^{-1})$;
- (3) $h_{prod}(A) = \sum_i \log_2(\sum_j a_{ij})$;
- (4) $H_{prod}(A) = h_{prod}(A) + h_{prod}(A^{-1})$.

² i_d or j_d may equal to 0, or $\tilde{E}_1^d, \dots, \tilde{E}_{i_d-1}^d, \tilde{E}_{i_d}^d, F_{j_d}^d, F_{j_d-1}^d, \dots, F_1^d$ can act in parallel.

These four cost functions roughly estimate the cost of decomposition through the sparsity of a matrix A , and reach their minimum when A is a permutation matrix. Therefore, they can guide the search in the cost-minimization process. h_{sum} is a rough estimation since there can be too many operations which lead to the same cost, and the remaining cost functions have two major improvements over h_{sum} . On the one hand, the inverse of the current matrix is taken into account, which was first proposed in [47]. Since a row operation on A is equivalent to a corresponding column operation on A^{-1} , adding the cost of the inverse matrix can provide a more balanced estimation of the distance to a permutation matrix. On the other hand, the logarithm of every row's Hamming weight is taken into account, which was first presented in [13]. h_{prod} gives priority to "almost done" rows such that the overall efficiency of elimination is guaranteed. Note that in cost-minimization algorithms one may end up with a sparse matrix where the rows and columns have few nonzero entries with common indices. This type of matrix represents a local minima from which it might be difficult to escape. Both these two considerations can help to avoid getting stuck in a local minima and can lead to better results.

According to their experiments with random matrices, this depth-oriented greedy method behaves well for small n (roughly $n < 40$). When n is larger, it performs worse than their block algorithm and often falls into a local minima. Their block algorithm [18] and Jiang *et al.*'s algorithm [27] have asymptotic optimal bounds and can handle larger matrices better.

4 Our Method and Its Applications

In this section, we first propose an improved greedy algorithm for finding low-depth CNOT circuits. Then we apply our algorithm to different linear building blocks with sizes of 16×16 and 32×32 that have been studied in [61]. Since the original greedy method in [18] is not applied to these matrices, we also apply their method to these matrices for comparison.

4.1 Our Method

Our method is based on the framework of de Brugière *et al.*'s greedy method [18], and differs from it in three aspects. First, in addition to considering the logarithm of each row's Hamming weight, we also consider the *square* of each row's Hamming weight. Second, we treat two cases of row and column operations differently when evaluating the cost, that is, each column's Hamming weight is considered when column operations are performed. Finally, we add a judgement of whether the current matrix can be implemented with depth 1 to better handle sparse matrices.

We first make an intrinsic observation about the problem of synthesizing low-depth CNOT circuits. Though not strict, the larger Hamming weight of a row i , the more potential gates need to be done on the row i . Therefore, prioritizing the rows or columns with larger Hamming weights might be a preferable choice

to obtain lower circuit depth. Intuited by this, we propose a new cost function h_{sq} which is based on the square of every row's Hamming weight:

$$h_{sq}(A) = \sum_i \left(\sum_j a_{ij} \right)^2.$$

We also notice that focusing on the Hamming weight of the rows ignores the effect of the column operations. So we propose two cost functions H_{sqr}, H_{sqc} which are based on h_{sq} to evaluate row operations and column operations respectively:

$$\begin{aligned} H_{sqr}(A) &= h_{sq}(A) + h_{sq}((A^{-1})^T); \\ H_{sqc}(A) &= h_{sq}(A^T) + h_{sq}(A^{-1}). \end{aligned}$$

Note that the cost of the corresponding transformation of the inverse matrix is under consideration. In our method, the cost after row operations is evaluated by $H_{sqr}(A)$, and the cost after column operations is evaluated by $H_{sqc}(A)$. In addition, the cost of the current matrix is defined as the maximum evaluation of H_{sqr} and H_{sqc} to explore more possibilities.

According to our experiments, using row and column cost functions based on h_{prod} will sometimes yield better results than using row and column cost functions based on h_{sq} , which means that h_{sq} is not the best choice for all matrices. So in practice, we also use cost functions H_{prodr} and H_{prodc} defined as follows:

$$\begin{aligned} H_{prodr}(A) &= h_{prod}(A) + h_{prod}((A^{-1})^T); \\ H_{prodc}(A) &= h_{prod}(A^T) + h_{prod}(A^{-1}). \end{aligned}$$

In our algorithm, we adopt a hybrid strategy of randomly using H_{prodr}, H_{prodc} or H_{sqr}, H_{sqc} , since both cases are likely to give the best result.

In addition, we give an equivalent condition to test whether a matrix can have depth 1. This helps determine whether the last searched L_r and L_c can be implemented in parallel, since the original algorithm can only find this better case with probability $\frac{2}{2^t}$, where t is the total number of CNOT gates in the last searched L_r and L_c .

Theorem 3. *Suppose the implementation of a permutation matrix is free. Given an invertible matrix $A_{n \times n}$ on \mathbb{F}_2 , A can be implemented with depth 1 if and only if the following conditions hold:*

- (a) *The Hamming weights of all A 's rows are less than or equal to 2.*
- (b) *For any two rows i, j of A with Hamming weight 2, $a_{ik}a_{jk} = 0$ for all k .*

Proof. It is easy to see that a set of row operations on a permutation matrix can be interpreted as a set of column operations and vice versa. So we only need to consider the parallelism of row operations to reduce a matrix to a permutation matrix.

Necessity. It is easy to see that target rows of type-3 matrices have Hamming weight 2 and other rows have Hamming weight 1. (b) holds since the reduced matrix is a permutation matrix.

Sufficiency. Without loss of generality, assume that the i -th row has Hamming weight 2 for $0 \leq i < l$, and other rows have Hamming weight 1. For each i such that $a_{i,p_i} = 1, a_{i,q_i} = 1$, there exists only one corresponding row t_i with Hamming weight 1 such that either $a_{t_i,p_i} = 1$ or $a_{t_i,q_i} = 1$, since A is invertible. Applying $E(t_i, i)$ for $0 \leq i < l$ gives the implementation with depth 1. \square

Detailed steps of our improved greedy algorithm are illustrated as Algorithm 1. The running time of this algorithm is dominated by evaluating the cost of all possible row or column operations. Suppose the considered matrix A is n by n . Ignoring the limitation of parallelism, at most n^2 row operations and n^2 column operations need to be evaluated. Since A^{-1} can be computed first and updated according to the corresponding operations on A , the cost of the resulting matrix can be computed in $O(n)$ times based on the cost of A . Therefore, the complexity of determining of an operation to be done for a current matrix is upper bounded by $O(n^3)$. Similar to de Brugière *et al.*'s greedy method, our algorithm behaves well for small scale matrices (roughly $n < 40$), and our often falls into a local minima when n is larger. Our algorithm is repeated tens of thousands of time for a matrix and the best result is recorded.

4.2 Application to AES MixColumns

We first focus on CNOT circuits of AES MixColumns. Previous researchers synthesized quantum circuits of AES MixColumns with different methods. Grassal *et al.* found one circuit with depth 39 by the LUP decomposition method [20]. Zou *et al.* adopted the same circuit, while Jaque *et al.* obtained a circuit with depth 111^3 [24]. Xiang *et al.*'s reduction framework produced a circuit with 92 CNOT gates and depth 41 [59], while Q#'s estimation of this circuit is 30, and Zhu *et al.* studied the exchange-equivalent sequence of this circuit [61] and obtained a new circuit with depth 28. Recently Liu *et al.* proposed a method for computing the depth of quantum circuits and then used it to evaluate many circuits generated by Xiang *et al.*'s method. They obtained a circuit with 98 CNOT gates and depth 16 [38]. If ancilla qubits are allowed, some CNOT circuits with low depth can be designed by converting optimized classical circuits (see [8, 32, 39] for an incomplete list) into quantum style. For example, Liu *et al.* converted the classical circuit in [32] with 105 XOR gates and classical depth 3 into quantum style with depth 11 and 105 ancilla qubits. Jang *et al.* optimized the circuit built upon the work in [32] and obtained an out-of-place circuit with 64 ancilla qubits and depth 8 [23].

³ In the Eurocrypt'20 paper [24], the authors remarked that they could not reproduce the result although they used the same technique. The authors of [23] analyzed that the reason may come from the encoding issue.

Algorithm 1. Improved greedy algorithm for synthesizing low-depth CNOT circuits.

Input: An invertible matrix $A_{n \times n}$

Output: A depth d and a vector of layers $Layers$ that implement A with length d .

$Layers \leftarrow \emptyset, Layers_r \leftarrow \emptyset, Layers_c \leftarrow \emptyset;$

$L_r \leftarrow \emptyset, L_c \leftarrow \emptyset;$

$List \leftarrow \emptyset;$

$B \leftarrow A;$

Randomly determine $H_r, H_c \leftarrow H_{sqr}, H_{sqc}$ or H_{prodr}, H_{prodc} .

$cost \leftarrow \max\{H_r(B), H_c(B)\};$

$can_one \leftarrow \text{False};$

▷ If the current matrix can be implemented with depth 1.

$d \leftarrow 0;$

while True do

$cost \leftarrow \max\{H_r(B), H_c(B)\};$

$mincost$ ← the minimum resulting cost of all available row operations (can act in parallel with those in L_r) of adding the i -th row to the j -th row of B (denoted $\{i, j, 0\}$), and if not can_one , all available column operations (can act in parallel with those in L_c) adding the i -th column to the j -th column of B (denoted $\{i, j, 1\}$); ▷ If the current matrix can be implemented with depth 1, only row operations need to be considered.

$List \leftarrow \{\text{All operations which can lead to the cost of the resulting matrix being } mincost\};$

if $mincost == cost$ **then**

if not can_one and $\text{Can-depth-one}(A)$ **then**

$can_one \leftarrow \text{True};$ ▷ If so, only row operations are considered in the next iteration.

end if

if $L_r.size()$ **then**

$d \leftarrow d + 1, Layers_r.push_back(L_r), L_r.clear();$

end if

if $L_c.size()$ **then**

$d \leftarrow d + 1, Layers_c.push_back(L_c), L_c.clear();$

end if

if $cost = 2n$ **then**

break;

end if

if $d >= 100$ **then**

return $d, \emptyset;$

▷ Too large d means the matrix may fall into a local minima.

end if

else

 Randomly choose one operation $\{i, j, op\}$ that minimizes the cost function of the resulting matrix, add $\{i, j, op\}$ to L_r if $op == 0$, or L_c if $op == 1$;

$B \leftarrow E(j+i)^{1-op} B E(i+j)^{op};$

end if

$List.clear();$

end while

Record a permutation P , satisfying $P(i) == j$ if $B[i][j] == 1$;

for i from 0 to $(Layer_c.size() - 1)$ **do**

$l \leftarrow \emptyset;$

for j from 0 to $(Layer_c[i].size() - 1)$ **do**

$\{t, c, op\} \leftarrow Layer_c[i][j], l.push_back(\{c, t\});$

end for

$Layers.push_back(l);$

end for

for i from $(Layer_r.size() - 1)$ down to 0 **do**

$l \leftarrow \emptyset;$

for j from 0 to $(Layer_r[i].size() - 1)$ **do**

$\{c, t, op\} \leftarrow Layer_c[i][j], l.push_back(\{P[c], P[t]\});$

end for

end for

return $d, Layers;$

We observe that related works of providing CNOT circuits of AES MixColumns either adopt non-depth-oriented search methods or determine the depth based on existing circuits. Instead, we use depth-oriented search algorithms to generate low-depth CNOT circuits with no ancilla qubits. We first apply the greedy algorithm in [18] with cost function H_{prod} to AES MixColumns and obtain a circuit with depth 12 and 128 gates. Then our improved greedy algorithm finds a circuit with depth 10 and 131 gates. It can be used to reduce the full depth of quantum circuits of AES without increasing the circuit width. The comparison with previous results is shown in Table 2.

Table 2. Comparison of CNOT circuits of the AES MixColumns matrix.

Source	# CNOT	W	FD
[8, 39]	206	135	13
[32]	210	137	11
[23]	169	96	8
[24]	277	32	111
[20, 63]	277	32	39
[59]	92	32	30
[61]	92	32	28
[38]	98	32	16
[18]	128	32	12
This paper	131	32	10

4.3 Applications to Many Proposed Matrices

Following the work of [61], we apply our method to various matrices in the literature including:

- some matrices used in block ciphers [2, 6, 7, 9–12, 15, 16, 25, 28, 48, 52];
- some MDS matrices which are constructed in [14, 26, 33, 37, 45, 46, 54].

Based on the CNOT circuits of these matrices provided by Xiang *et al.*'s method, Zhu *et al.* evaluated their move-equivalent sequence depth by Q# [43], and investigated their exchange-equivalent sequences. Except for a few small-scale matrices, we can find much better results with lower depths for these matrices. For the matrices used in block ciphers, we have succeeded in reducing the circuit depth for all of them except for a few matrices that already have CNOT circuits with small depth (see Table 3). For the many constructed MDS

matrices, we can optimize the depth of CNOT circuits for all of them (see Table 4). Overall, our improved greedy algorithm gives the best results with the lowest depth for all of the matrices⁴.

Table 3. Comparison of the depth/gate count of CNOT circuits for matrices used in block ciphers.

Cipher	Size	Q#	[61]	This paper	[18]
AES ^a [16]	32	30/92	28/92	10 /131	12/128
ANUBIS [10]	32	26/98	20/98	10 /119	14/136
CLEFIA M0 [52]	32	30/98	27/98	10 /110	13/126
CLEFIA M1 [52]	32	21/103	16/103	10 /128	13/127
FOX MU4 [28]	32	55/136	48/136	21 /265	21 /200
QARMA128 [6]	32	6/48	5/48	3 /48	3 /48
TWOFISH [48]	32	37/111	29/111	15 /175	18/187
WHIRLWIND M0 [9]	32	65/183	51/183	28 /331	28 /286
WHIRLWIND M1 [9]	32	69/190	54/190	22 /290	25/279
JOLTIK [25]	16	20/44	17/44	7 /52	9/48
MIDORI [7]	16	3 /24	3 /24	3 /24	3 /24
SmallScale AES [15]	16	20/43	19/43	10 /62	11/59
PRIDE L0 [2]	16	3 /24	3 /24	3 /24	3 /24
PRIDE L1 [2]	16	5/24	5/24	3 /24	3 /24
PRIDE L2 [2]	16	5/24	5/24	3 /24	3 /24
PRIDE L3 [2]	16	6/24	6/24	3 /24	3 /24
PRINCE M0 [12]	16	6/24	6/24	3 /24	3 /24
PRINCE M1 [12]	16	6/24	6/24	3 /24	3 /24
QARMA64 [6]	16	6/24	5/24	3 /24	3 /24
SKINNY [11]	16	3 /12	3 /12	3 /12	3 /12

^a A recent result of 16/98 is given in [38]

⁴ Note that for a few matrices, implementations with the same depth but fewer gates can be searched using de Brugière *et al.*'s algorithm. Therefore, their algorithm could be used in combination with our algorithm in order to search for better low depth CNOT circuits.

Table 4. Comparison of the depth/gate count of CNOT circuits for many constructed MDS matrices.

Matrices	Size	Move-eq	[61]	This paper	[18]
4×4 matrices in $\text{GF}(4, \mathbb{F}_2)$					
[14]	16	23/41	21/41	10/59	12/57
[26]	16	24/41	18/41	9/49	9/48
[37]	16	27/41	26/41	11/63	12/65
[54]	16	25/44	22/44	11/59	11/59
[33]	16	29/44	27/44	11/62	12/65
[26] (Involutory)	16	15/41	14/41	9/54	13/54
[54] (Involutory)	16	19/44	16/44	7/52	9/48
[33] (Involutory)	16	27/44	25/44	7/52	9/48
[45] (Involutory)	16	12/38	11/38	8/46	8/44
4×4 matrices in $\text{GF}(8, \mathbb{F}_2)$					
[14]	32	56/144	47/144	18/208	20/188
[26]	32	26/82	22/82	9/100	9/96
[37]	32	67/121	54/121	21/235	23/203
[33]	32	55/104	42/104	13/164	16/167
[54]	32	23/90	20/90	10/112	11/118
[45]	32	47/114	40/114	20/218	20/190
[26] (Involutory)	32	18/83	14/83	9/102	13/108
[54] (Involutory)	32	18/91	16/91	8/101	9/96
[33] (Involutory)	32	19/87	19/87	8/99	8/98
[45] (Involutory)	32	19/93	18/93	10/121	12/119
8×8 matrices in $\text{GF}(4, \mathbb{F}_2)$					
[46]	32	54/183	44/183	29/351	33/302
[54]	32	59/170	49/170	28/349	29/286
[54] (Involutory)	32	47/185	37/185	29/337	30/300
8×8 matrices in $\text{GF}(8, \mathbb{F}_2)$					
[54] (Involutory)	64	50/348	37/348	22/484	25/412

5 The Compressed Pipeline Structure for Iterative Primitives

In this section, we first introduce some structures used for quantum circuits of AES in previous work. Then we propose a new structure named compressed pipeline structure. Finally we make some comparisons in different levels and introduce its application in the Grover oracle and the Encryption oracle.

5.1 Existing Structures

Many structures have been proposed to synthesize quantum circuits of AES. Some of them are based on out-of-place round functions, including the pipeline structure \mathcal{S}_p , the zig-zag structure \mathcal{S}_z and the out-of-place based (OP-based in short) round-in-place structure \mathcal{S}_i . These structures treat the round function and its inverse as a unit. Therefore, the metric “round depth” - the number of layers of the round function - is then used to describe the depth of a structure, and the width of a structure is closely related to the number of intermediate states and the number of parallel round functions.

The pipeline structure, which is first mentioned in reversible logic implementations of AES in [17], was proposed by Jaques *et al.* in [24]. It has low round depth and large width and is used to construct low T -depth quantum circuits of AES in [22–24, 38]. The zig-zag structure was first put forward by Grassal *et al.* in [20] to reduce the number of intermediate states. It is used to construct low-width circuits of AES in [3, 20, 30]. To further reduce the number of intermediate states, Zou *et al.* presented the improved zig-zag structure [63], and Huang *et al.* proposed the OP-based round-in-place structure to construct in-place circuits on the basis of out-of-place circuits. Denote the j -th round function \mathcal{R}_j which satisfies $\mathcal{R}_j(c_{j-1}) = c_j$, then the out-of-place oracle $\mathcal{O}_{\mathcal{R}_j}$ takes $|x\rangle |y\rangle$ as input and outputs $|x\rangle |y \oplus \mathcal{R}_j(x)\rangle$. The input register and output register is distinguished by notation $\mathcal{R}_{j,i}$ and $\mathcal{R}_{j,o}$ respectively. For simplicity, the key schedule and the ancilla qubits are omitted. \mathcal{S}_p and \mathcal{S}_z compute the desired output $|c\rangle$ along with some redundant states, as illustrated in Fig. 3, 4, respectively. The construction of the in-place function in \mathcal{S}_i is shown in Fig. 5.

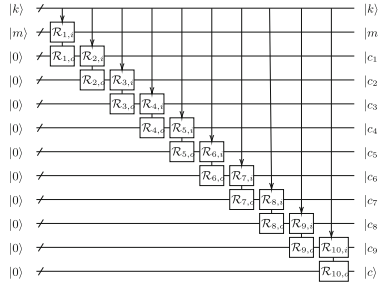


Fig. 3. The pipeline structure \mathcal{S}_p .

There are also some other structures which do not take the out-of-place round function as a unit. On the one hand, an in-place round function can be directly designed without out-of-place round functions. For example, Li *et al.* proposed an in-place quantum circuit of AES S-box with only 8 ancilla qubits, and then used it to synthesize a quantum circuit of AES under the straight-line structure, which has the lowest width to date [35]. On the other hand, some out-of-place round functions themselves can be decomposed into computation and

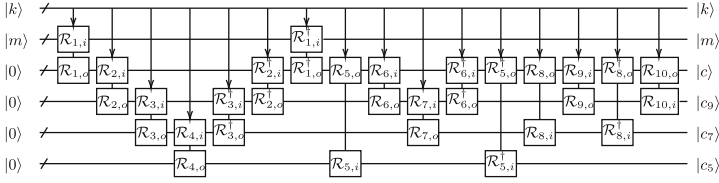


Fig. 4. The zig-zag structure \mathcal{S}_z .

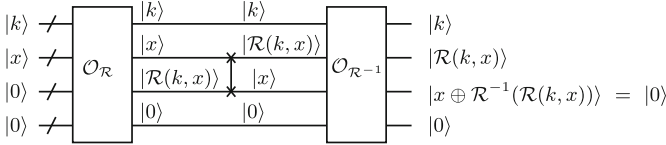


Fig. 5. OP-based round-in-place function in \mathcal{S}_i .

uncomputation. The shallowed pipeline structure proposed by Jang *et al.* delays the uncomputation of one round function (if exists) to the next round in the pipeline structure to reduce the full depth [23]. Liu *et al.* improved this structure by sharing the ancilla qubits of the computation and uncomputation to save qubits [38].

5.2 Compressed Pipeline Structure

In this section we propose the compressed pipeline structure.

We first make some observations on existing structures. Though the pipeline structure has the lowest round depth, too many qubits store the intermediate states. At the same time, the zig-zag structure and its improvements clean some intermediate states to save qubits, but at a cost of almost twice the round depth of the pipeline structure. To combine the advantages of the above two structures, we propose a strategy of computing new states and eliminating intermediate states in parallel.

Specifically, when c_{j+1} ($j \geq 1$) is generated, the register storing c_{j-1} can be cleaned by $\mathcal{O}_{\mathcal{R}_{j-1}^{-1}}$ in parallel and then can be reallocated in further use. In fact, our structure can be thought of as adding the clean process to the pipeline structure, hence the name compressed pipeline structure denoted by \mathcal{S}_{cp} . Since the input register of both $\mathcal{O}_{\mathcal{R}_{j-1}^{-1}}$ and $\mathcal{O}_{\mathcal{R}_j}$ is the same, $|c_j\rangle$ is copied by CNOT gates so that $\mathcal{O}_{\mathcal{R}_{j-1}^{-1}}$ and $\mathcal{O}_{\mathcal{R}_j}$ can act in parallel. Therefore, our structure requires four intermediate message registers, while a round function and its inverse function need to be executed in parallel. \mathcal{S}_{cp} is illustrated in Fig. 6.

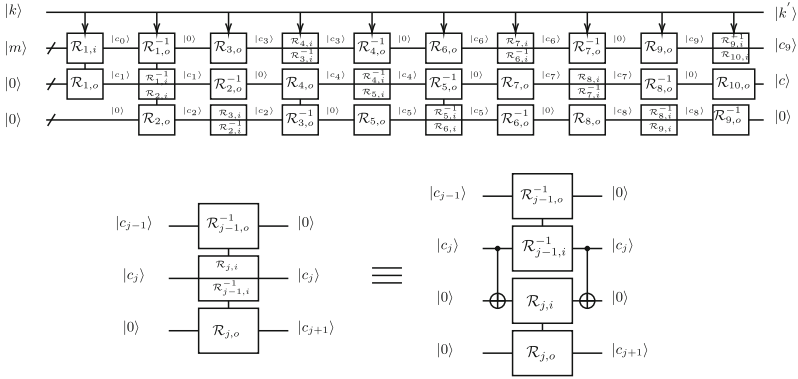


Fig. 6. The compressed pipeline structure \mathcal{S}_{cp} . For convenience, the copy of the $|c_j\rangle$ state is simplified as “split into two parts”.

5.3 Comparison of Different Structures

In this subsection, we will compare the round depth and width of different structures that takes the round function as a unit and illustrate the use of our \mathcal{S}_{cp} in two scenarios: the Grover oracle and the Encryption oracle.

We first clarify some parameters for the different structures. Suppose $\mathcal{O}_{\mathcal{R}_j}$ has round depth 1 and needs α ancilla qubits. Since the components of $\mathcal{O}_{\mathcal{R}_j}$, $\mathcal{O}_{\mathcal{R}_j^{-1}}$ are almost the same, it is reasonable to regard them as having the same cost. Suppose the round function iterates for r rounds, and one message register needs n qubits. The width of the key schedule is set to k' to show the difference from the other structures, because in our structure, the parallel execution of a round function with its inverse means that two consecutive roundkeys are required.

The comparison with previous structures under the above parameters is outlined in Table 5. Since we treat the round function as a unit, only the comparison between $\mathcal{S}_p, \mathcal{S}_z, \mathcal{S}_i$ with our \mathcal{S}_{cp} are considered.

Table 5. The comparison of different structures, where t is the minimal number such that $\sum_{i=1}^t i > r$.

Structure	Round depth	Width
\mathcal{S}_p	r	$k + (r + 1)n + \alpha$
\mathcal{S}_z	$\approx 2r$	$k + tn + \alpha \approx k + \sqrt{2rn} + \alpha$
\mathcal{S}_i	$2r$	$k + 2n + \alpha$
This paper	r	$k' + 4n + 2\alpha$

Our \mathcal{S}_{cp} has the same round depth r as \mathcal{S}_p , and at the same time needs 4 message registers instead of $r + 1$ message registers in \mathcal{S}_p . Therefore, our \mathcal{S}_{cp} will

have lower width than \mathcal{S}_p and lower $D-W$ cost than $\mathcal{S}_z, \mathcal{S}_i$ if α and k' are small enough.

Circuits for Grover Oracles. We first consider the Grover oracle: $|y\rangle|q\rangle \rightarrow |y\rangle|q \oplus f(y)\rangle$, where $f(y)$ is a boolean function which outputs one bit 1 or 0. An exhaustive key search Grover oracle has input state $|k\rangle$, and the correctness of the key is verified by some plaintext-ciphertext pairs. For simplicity we consider the case of one pair (m_0, c_0) . Denote an encryption circuit as C_* , the Grover oracle works as follows:

- C_* computes $|c\rangle$ with $|m_0\rangle$ and $|k\rangle$.
- A comparison process compares $|c\rangle$ with $|c_0\rangle$ to decide whether to flip $|q\rangle$.
- Do the uncomputation with C_*^\dagger .

It can be seen that the cost of the Grover oracle is almost twice that of the encryption circuit. Therefore, the cost of Grover oracles with different structures can be evaluated directly by referring to the cost of different encryption circuits in Table 5.

Since uncomputation of recovering m is necessary in the Grover oracle, the depth of the Grover oracle is dominated by that of C_* . Thus, \mathcal{S}_p is used to construct low-depth circuits of the Grover oracle in related research. Our circuit \mathcal{S}_{cp} greatly reduces the use of message registers to store intermediate states and will have lower width than \mathcal{S}_p if the number of ancilla qubits required in round functions is small enough.

Circuits for Encryption Oracles. We then consider the Encryption oracle defined in [29]: $|m\rangle|0\rangle \rightarrow |m\rangle|Enc(m)\rangle$, where m is the plaintext and $Enc(m)$ is the encryption of m with a pre-fixed key. Encryption oracles allows the input register to be in a superposition $\sum_m |m\rangle$, and then the output will be a superposition $\sum_m |m\rangle|Enc(m)\rangle$. Note that the key register is not needed in the Encryption oracle since the roundkeys can be precomputed classically and the AddRound-Key can be realized by applying X gates on specified qubits. The construction of Encryption oracles using $\mathcal{S}_p, \mathcal{S}_z$ or \mathcal{S}_i is shown in Fig. 7.

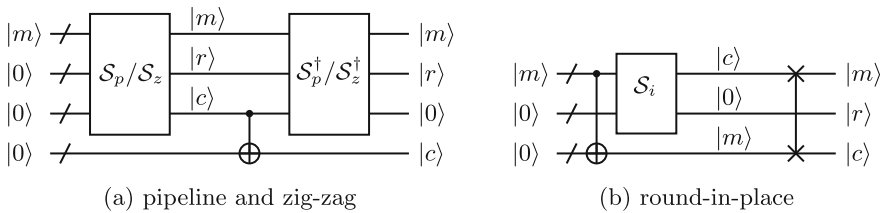


Fig. 7. The Encryption oracles based on different structures

We show that our structure \mathcal{S}_{cp} can be used to construct an Encryption oracle that has a smaller round depth. The round depth of an Encryption oracle using

\mathcal{S}_i is $2r$ since there is no redundant intermediate states, and the round depth of an Encryption oracle using $\mathcal{S}_p/\mathcal{S}_z$ is twice the round depth of $\mathcal{S}_p/\mathcal{S}_z$ since uncomputation is needed to clean redundant states. However, the advantage of our structure is that \mathcal{S}_{cp} can greatly reduce the cost for cleaning redundant states. Since $|c_{j-1}\rangle$ is cleaned by $|c_j\rangle$, the remaining redundant states contains only $|c_{r-1}\rangle$, which can be cleaned by $|c\rangle$ using \mathcal{R}_r^{-1} . Thus, the round depth of an Encryption oracle using \mathcal{S}_{cp} and \mathcal{R}_r^{-1} is only $r + 1$, almost half the previous record. The comparison with previous results is outlined in Table 6.

Table 6. The depth and width of Encryption oracles with different structures

Encryption oracle	\mathcal{S}_p	\mathcal{S}_z	\mathcal{S}_i	This paper
round depth	$2r$	$\approx 4r$	$2r$	$r + 1$
width	$(r + 1)n + \alpha$	$\approx \sqrt{2rn} + \alpha$	$(1 + 2)n + \alpha$	$(1 + 4)n + 2\alpha$

6 Quantum Circuits of AES

In this section we give several kinds of detailed quantum circuits of AES under the guidance of our structure in Sect. 5, including an Encryption oracle, encryption circuits under our structure and improved encryption circuits under the shallowed pipeline structure. The depth 10 circuit of AES MixColumns in Sect. 4 are used in all circuits to reduce the full depth, and the different circuits for AES S-box are introduced in Subsect. 6.1. Encryption circuits for the encryption part and the key schedule of AES-128 are specified in Subsect. 6.2, and the cost is compared with other circuits in different cases in Subsect. 6.3⁵. In Subsect. 6.4 we synthesize an Encryption oracle of AES-128 with the lowest T -depth to date, and in Subsect. 6.5 we give an improved encryption circuit under the shallowed pipeline structure with the lowest $TofD-W$ cost to date.

6.1 Quantum Circuits of AES S-Box

We first introduce some knowledge on quantum circuits of AES S-box. The C_2 circuit of AES S-box defined by $C_2 : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus S(x)\rangle$ is the main concern for out-of-place implementations, since C_1 circuits defined by $C_1 : |x\rangle |0\rangle \mapsto |x\rangle |S(x)\rangle$ are special cases of C_2 circuits and are easier to design. The C_3 circuit defined by $C_3 : |S(x)\rangle |x\rangle \mapsto |S(x)\rangle |0\rangle$ can be efficiently constructed on a C_1 circuit with a few more CNOT gates by the method in [22]. Moreover, some C_2 circuits of AES S-box can be decomposed into Sbox and SubS[†]. The Sbox circuit

⁵ The main difference between the encryption circuit of AES-128, AES-192 and AES-256 lies in the key schedule. The key schedules and the cost analysis for encryption circuits of AES-192 and AES-256 are presented in Appendix B.

is defined by Sbox: $|x\rangle|0\rangle|y\rangle \mapsto |x'\rangle|r\rangle|y \oplus S(x)\rangle$, where $|r\rangle$ is the redundant state. Sbox can be decomposed into two parts denoted by SubS and SubC. SubS takes $|x\rangle|0\rangle$ as input and outputs $|x'\rangle|r\rangle$, where $|r\rangle$ contains linear components of $S(x)$. Then SubC adds them to $|y\rangle$ to get $|y \oplus S(x)\rangle$.

Many researchers have studied low Toffoli-depth Sboxes, which is illustrated in Table 7. In these related works, the target qubit of each Toffoli gate in SubS is always a new qubit $|0\rangle$, so the Toffoli gates can be replaced by qAND gates with T -depth 1, and SubS[†] can be realized with T -depth 0 using qAND[†]. Therefore, these related works can be used to synthesize low T -depth qAND-based C_2 circuits of AES S-box.

Note that the input register of some Sboxes can remain unchanged while others cannot. An Sbox is defined to be input-invariant if it can keep the input register unchanged, which means the information of the input register $|x\rangle$ is not lost before SubS[†] or Sbox[†] is done. We observe that, the reason why some Sboxes are not input-invariant is that the input register is updated by some ancilla qubits with CNOT gates to save qubits, and these ancilla qubits themselves are also updated. It is worthy to note that the updating of $|x\rangle$ can be uncomputed with only CNOT gates, as the target qubit of all Toffoli gates is always a new qubit $|0\rangle$ in related works of low Toffoli-depth Sboxes. Therefore, adding some CNOT gates can make this kind of Sbox input-invariant without increasing the number of ancilla qubits and the Toffoli-depth (see Table 7 for our results). The full depth is also not increased when the Toffoli gates are decomposed.

Table 7. Some low $TofD$ Sboxes

Source	#CNOT	#1qClifford	#Toffoli	$TofD$	Ancilla qubits	Input-invariant
[24]	186	4	34	6	120	✓
[22]	214	4	34	4	120	✓
[22]	356	4	78	3	182	✓
[38]	168	4	34	4	74	✗
This paper	179	4	34	4	74	✓
[38]	196	4	34	4	60	✗
This paper	207	4	34	4	60	✓
[23]	313	4	78	3	136	✗ ^b
[23]	162	4	34	4	68 ^a	✗ ^b

^a The full depth of this circuit is smaller when the Toffoli gates are decomposed.

^b Since the authors do not give specific implementations, we cannot give detailed costs for their input-invariant versions

The process of finding a sequence of CNOT gates added to a given low Toffoli-depth Sbox to make it input-invariant can be integrated into an algorithm. It involves recording the qubits whose updatings should be memorized. All the input qubits are recorded at the beginning, and each qubit used to update a

recorded qubit is recorded. Finally, a sequence of uncomputing the memorized updatings is returned. The detailed process is illustrated in Algorithm 2. Note that in order not to increase the Toffoli-depth, this algorithm is only suitable for such low Toffoli-depth Sboxes where the updating of input qubits is related with only CNOT gates.

Some C_1 circuits for low-width Toffoli-based circuits are also studied, see Table 8 for some recent works. If these C_1 circuits are used for AES S-box, our encryption circuit of AES-128 in Subsect. 6.3 will have the lowest $ToFD-W/TD-W$ cost compared to other structures which take the round function as a unit.

Table 8. Some Toffoli-based C_1 circuits of AES S-box

Source	#CNOT	#1qClifford	#Toffoli	Toffoli-depth	Ancilla qubits
[36]	193	4	57	24	5
[36]	195	4	57	22	6
[35]	197	4	44	32	4

Algorithm 2 Make an Sbox input-invariant.

Input: An NCT-based circuit $\mathcal{C} = \{g_0g_1 \dots g_{t-1}\}$ of an Sbox with input qubits $|x_0x_1 \dots x_7\rangle$, ancilla qubits $|r_0r_1 \dots r_{m-1}\rangle$ and output qubits $|y_0y_1 \dots y_7\rangle$. The updating of input qubits in the input Sbox should be related with only CNOT gates.

Output: A sequence of CNOT gates which is added to the Sbox to make it input-invariant.

```

seq ← [];
for i from 0 to m - 1 do
    updated[i] ← 0;
end for
for i from 0 to t - 1 do
    if  $g_i$  is not a CNOT gate then
        Continue;
    end if
    Let  $a$  be the control qubit and  $b$  be the target qubit of  $g_i$ .
    if  $b$  is some  $|x_i\rangle$  or some  $|r_j\rangle$  with  $updated[j] = 1$  then
        seq.append( $g_i$ );
        if  $a$  is some  $|r_j\rangle$  then
            updated[j] ← 1;
        end if
    end if
end for
seq.reverse();
return seq;

```

6.2 Round Function and Key Schedule

In this subsection we give the detailed circuits of the iterative functions that we define for the encryption circuit and key schedule of AES-128. For AES-192 and AES-256, the main difference lies in the key schedule, which is illustrated in Appendix B.

For the encryption circuit, we define the beginning function B and the j -round function F_j which are shown in Fig. 8. For simplicity, the result of applying multiple AES S-boxes on a qubit register $|x\rangle$ is denoted by $|S(x)\rangle$ throughout the rest of the paper. B and F_j acts as follows:

$$\begin{aligned}
 B &: |m\rangle |0\rangle |0\rangle |0\rangle \mapsto |c_0\rangle |S(c_0)\rangle |c_1\rangle |0\rangle, \\
 F_j &: |c_{j-1}\rangle |S(c_{j-1})\rangle |c_j\rangle |0\rangle \mapsto |c_j\rangle |S(c_j)\rangle |c_{j+1}\rangle |0\rangle.
 \end{aligned}
 \tag{3}$$

As a result, our encryption circuit is synthesized by connecting B, F_1, F_2, \dots, F_9 . Note that C_{cp} does not strictly adhere to the structure \mathcal{S}_{cp} in Fig. 6. One can see that outputs of AES S-box are copied for cleaning the inputs in the next round, which saves the cost of the inverse of linear layers, and that our circuit C_{cp} is more compact, has clear linear and nonlinear layers, and has fewer linear layer components of AES than the circuit which adhere strictly to \mathcal{S}_{cp} . In a nonlinear layer, 16 C_1 circuits and 16 C_3 circuits are executed in parallel for the encryption circuit.

We then present a new key schedule circuit which is suitable with our new encryption circuit, since in F_j , two consecutive roundkeys $|k_j\rangle, |k_{j+1}\rangle$ should be able to be computed simultaneously by CNOT gates. Instead of storing $|k_j\rangle, |k_{j+1}\rangle$ in eight 32-qubit registers, we store linear components of two consecutive roundkeys registers to save qubits. The linear components of two consecutive roundkeys $|k_j\rangle, |k_{j+1}\rangle$ include $|k_j^0\rangle |k_j^1\rangle |k_j^2\rangle |k_j^3\rangle |S(k_j^3)\rangle$, and the computation of $|k_j\rangle, |k_{j+1}\rangle$ with CNOT gates is based on the dependence of consecutive

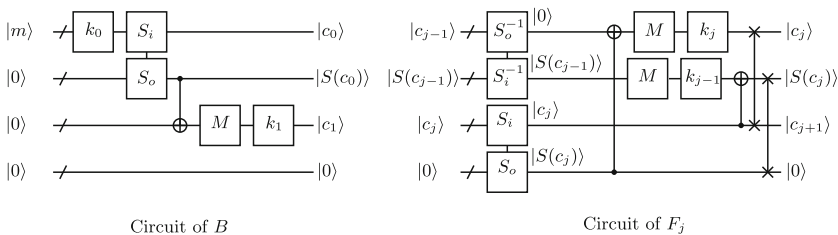


Fig. 8. Circuits of B and F_j . S_i, S_o and S_i^{-1}, S_o^{-1} stand for the input and output registers of C_1 circuits and C_3 circuits, respectively. MixColumns no longer acts on the first message register in F_9 . ShiftRows are omitted for simplicity throughout the rest of the paper.

roundkeys illustrated below:

$$\begin{cases} k_{j+1}^0 = Const_{j+1} \oplus S(k_3^j) \oplus k_j^0 \\ k_{j+1}^1 = Const_{j+1} \oplus S(k_3^j) \oplus k_j^0 \oplus k_j^1 \\ k_{j+1}^2 = Const_{j+1} \oplus S(k_3^j) \oplus k_j^0 \oplus k_j^1 \oplus k_j^2 \\ k_{j+1}^3 = Const_{j+1} \oplus S(k_3^j) \oplus k_j^0 \oplus k_j^1 \oplus k_j^2 \oplus k_j^3 \end{cases}, \quad (4)$$

where $Const_{j+1}$ is the $(j + 1)$ -th round constant.

By the dependence of consecutive roundkeys, we construct a circuit of key schedule which can compute linear components of two consecutive roundkey states with six 32-qubit registers. The beginning iteration K_0 and the j -th iteration K_j act as follows:

$$\begin{aligned} K_0 &: |k_0^0\rangle |k_0^1\rangle |k_0^2\rangle |k_0^3\rangle |0\rangle |0\rangle \mapsto |k_0^0\rangle |k_0^1\rangle |k_0^2\rangle |k_0^3\rangle |S(k_0^3)\rangle |0\rangle \\ K_j &: |k_{j-1}^0\rangle |k_{j-1}^1\rangle |k_{j-1}^2\rangle |k_{j-1}^3\rangle |S(k_{j-1}^3)\rangle |0\rangle \mapsto |k_j^0\rangle |k_j^1\rangle |k_j^2\rangle |k_j^3\rangle |S(k_j^3)\rangle |0\rangle. \end{aligned} \quad (5)$$

The circuit of K_j is illustrated in Fig. 9, and the circuit of K_0 is omitted due to its simplicity. In the nonlinear layer of K_j , 4 C_1 circuits and 4 reversed C_1 circuits are executed in parallel. Therefore, F_j and K_j can be synchronized. It is easy to see that $|k_j\rangle |k_{j+1}\rangle$ can be computed by the linear components with depth 5, thus a small increase in the depth of AddRoundKey trades for key register savings.

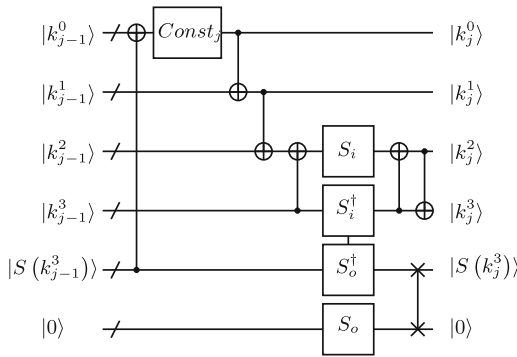


Fig. 9. The j -th iteration K_j of the key schedule.

Moreover, only five 32-qubit registers are enough if one uses the qAND-based Sbox, since Sbox[†] can clear the redundant states without increasing the T -depth. The beginning iteration K'_0 and the j -th iteration K'_j acts as follows:

$$\begin{aligned} K'_0 &: |k_0^0\rangle |k_0^1\rangle |k_0^2\rangle |k_0^3\rangle |0\rangle |0\rangle \mapsto |k_0^0\rangle |k_0^1\rangle |k_0^2\rangle |k_0^3\rangle |r_0\rangle |0\rangle \\ K'_j &: |k_{j-1}^0\rangle |k_{j-1}^1\rangle |k_{j-1}^2\rangle |k_{j-1}^3\rangle |r_{j-1}\rangle |0\rangle \mapsto |k_j^0\rangle |k_j^1\rangle |k_j^2\rangle |k_j^3\rangle |r_j\rangle |0\rangle \end{aligned} \quad (6)$$

The corresponding circuit K'_j is shown in Fig. 10. $|r_{j-1}\rangle$ and $|r_j\rangle$ are the redundant states within the computations of $|S(k_{j-1})\rangle$ and $|S(k_j)\rangle$, respectively. Similarly, K_0 's corresponding circuit K'_0 needs 4 Sboxes. It is worthy to note that the Sbox is actually input-invariant, which is easy to achieve based on our analysis in Subsect. 6.1.

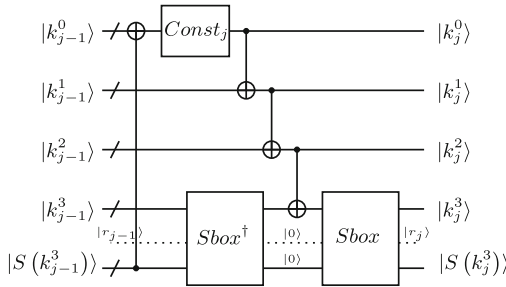


Fig. 10. K'_j with Sbox and Sbox[†]. The dashed line represents the ancilla qubits of qAND-based Sbox.

6.3 Encryption Circuits of AES-128

In this subsection we give detailed encryption circuits of AES-128, which can be constructed with the iterative circuits F_j, B, K_j, K'_j defined by us. F_j runs in parallel with B_j , where K_j add specific 32-qubit registers to the message registers by CNOT gates. See Fig. 11 for the hierarchy of our encryption circuit of AES-128.

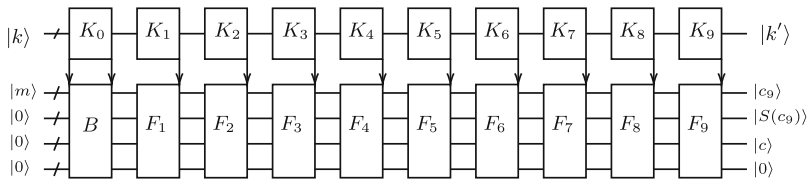


Fig. 11. Our encryption circuit of AES-128. The arrows indicate the AddRoundKey process at the beginning or end of K_j .

Denote our circuit under \mathcal{S}_{cp} as C_{cp} , and the circuit under \mathcal{S}_p and \mathcal{S}_i as C_p and C_i , respectively. We then compare the cost of our C_{cp} with C_p, C_i in both Toffoli-based and qAND-based AES S-box scenarios. For different circuits, the number of qubits required for the key registers and message registers, parallel

C_1, C_2 circuits⁶ and layers of AES S-box are shown in Table 9. One can see that our circuit C_{cp} do not need C_2 circuits.

Table 9. Costs of encryption circuits of AES-128 for different structures

Circuits	C_p	C_i	C_{cp} with K_j	C_{cp} with K'_j
Qubits of key registers	128	128	192	160
Qubits of message registers	128×11	128×2	128×4	128×4
C_1 circuits in parallel	16	16	40	36
C_2 circuits in parallel	4	2	0	0
Layers of AES S-box	10	20	10	10

The cost of different structures can be computed when the number of ancilla qubits and the Toffoli/ T -depth of AES S-box are determined. For simplicity of comparison, assume that both C_1 and C_2 circuits require m ancilla qubits. In case of using Toffoli-based AES S-box, our circuit needs fewer ancilla qubits than C_p when $m < 42$, and has lower $TofD-W$ cost than C_i when $m < 16$. So our circuit will have lower $TofD-W$ -cost with state-of-art low-width AES S-box. In case of using qAND-based Sbox, our circuit needs fewer ancilla qubits than C_p when $m < 54$, and has lower $TD-W$ cost than C_i for all $m > 0$. In conclusion, among the structures which take the round function as a unit, our compressed pipeline structure can provide different encryption circuits with better $TofD/TD-W$ trade-offs when the width of AES S-box is small. An encryption circuit using [36]’s low-width S-Box with more balanced $TofD$ and W is given in Table 10⁷.

Since a Grover oracle is composed of an encryption circuit, its dagger and a small comparison process, the choice of parameters to make the $TD-W$ cost of Grover search lower is basically consistent with the above analysis.

6.4 AES Encryption Oracle with Lower T -Depth

In this subsection we synthesize Encryption oracles of AES using the encryption part of our encryption circuit C_{cp} .

As introduced in Subsect. 5.3, previous researchers synthesized AES-128 Encryption oracles which cannot break the limit of 2×10 layers of AES S-box. Since the roundkeys can be precomputed in the Encryption oracle, the redundant states of our circuit C_{cp} only include $|c_9\rangle |S(c_9)\rangle$, which can be cleaned with only one layer of AES S-box. The clear function C shown in Fig. 12 takes

⁶ Since a C_3 circuit can be constructed by a C_1 circuit with a few more CNOT gates using the method in [22], we regard them as the same type of C_1 circuits.

⁷ So far the circuit under the improved compressed pipeline structure with our improvement of the input-invariant Sbox has the lowest $TofD-W$ cost, since Sbox[†] in the round function oracle is delayed to the next round and the technique of combined Sbox and Sbox[†] can save many qubits.

$|c_9\rangle |S(c_9)\rangle |c\rangle |0\rangle$ as input and outputs $|0\rangle |0\rangle |c\rangle |0\rangle$. Therefore, the AES-128 Encryption oracle can be constructed with $(10 + 1)$ layers of AES S-box.

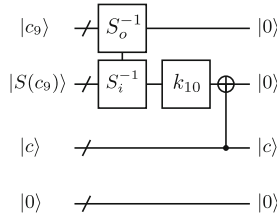


Fig. 12. The clear function C .

Using the qAND-based C_1 circuits and C_3 circuits with T -depth 3, we construct an AES-128 Encryption oracle with T -depth 33, which breaks the previous record of T -depth 60 in [22]. In the same way, an AES-192 (AES-256, respectively) Encryption oracle can be synthesized with T -depth 39 (45, respectively).

Since in AES-like Hashing the roundkeys are actually constants, our circuit can also be used to construct quantum oracles of AES-like Hashing with lower T -depth.

6.5 Key Schedule of the Shallowed Pipeline Structure with Lower Width

In this subsection we give an encryption circuit with the lowest $TofD-W$ cost to date under the shallowed pipeline structure. This encryption circuit uses an improved key schedule with our design of input-invariant Sbox in Subsect. 6.1.

Jang *et al.* proposed the shallowed pipeline structure which delays the cleaning of redundant states by SubS^\dagger to the next round to reduce the full depth [23]. Then, Liu *et al.* improved the structure by sharing the ancilla qubits in Sbox and SubS^\dagger [38] to save qubits. They both adopted the straight line structure for the key schedule, where $|k_{j-1}\rangle$ will be updated by $|k_j\rangle$ in the j -th round. As introduced in Subsect. 6.1, the Sbox given in [22] with unoptimized width is input-invariant, but the Sbox given in [38] with optimized width is no longer input-invariant, which leads to the lose of information in the input register. Since Sbox and SubS^\dagger need $|k_j^3\rangle$ and $|k_{j-1}^3\rangle$ for each j , respectively, Jang *et al.* allocated 32 additional qubits for storing $|k_{j-1}^3\rangle$ when using the input-invariant Sbox. Since the Sbox used by Liu *et al.* has smaller width but is no longer input-invariant, 10×32 qubits are allocated for storing all $|k_{j-1}^3\rangle$ with $1 \leq j \leq 10$. We show that the extra allocation of qubits for storing keywords is not necessary.

On the one hand, we have succeeded to make a low Toffoli-depth Sbox input-invariant with a few more CNOT gates in Subsect. 6.1. On the other hand, by the dependence of consecutive roundkeys in Eq. 4 used in K_j 's, we have $k_{j-1}^3 = k_j^2 \oplus k_j^3$. Therefore, unchanged $|k_j^3\rangle$ and the feasibility of computing $|k_{j-1}^3\rangle$

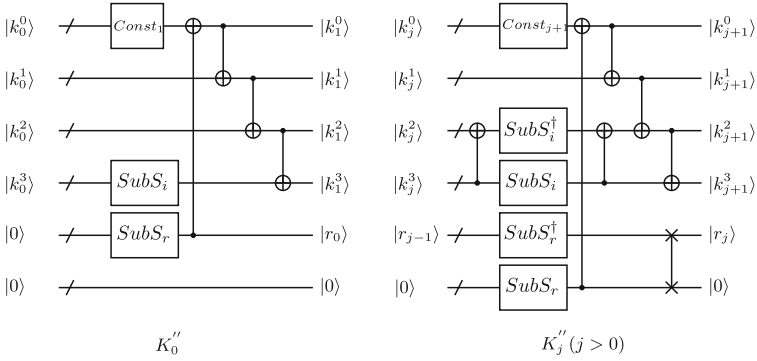


Fig. 13. K''_j for the shallowed pipeline structure. $SubS_i$ represents the input register of $SubS$, and $SubS_r$ represents the register that will store the redundant state.

with $|k_j^2\rangle, |k_j^3\rangle$ means that the allocation of extra qubits for the key schedule in [23, 38] is unnecessary. Our K''_j 's for the shallowed pipeline structure which are similar to K_j 's only need 128 qubits for the key registers and work as follows:

$$\begin{aligned}
 K''_0 : |k_0^0\rangle |k_0^1\rangle |k_0^2\rangle |k_0^3\rangle |0\rangle &\mapsto |k_1^0\rangle |k_1^1\rangle |k_1^2\rangle |k_1^3\rangle |r_0, 0\rangle. \\
 K''_j : |k_j^0\rangle |k_j^1\rangle |k_j^2\rangle |k_j^3\rangle |r_{j-1}, 0\rangle &\mapsto |k_{j+1}^0\rangle |k_{j+1}^1\rangle |k_{j+1}^2\rangle |k_{j+1}^3\rangle |r_j, 0\rangle.
 \end{aligned}
 \tag{7}$$

Using our K''_j and our input-invariant Sbox, we achieve an encryption circuit of AES-128 under the shallowed pipeline structure with the lowest *TofD-W* cost 130720 to date⁸. A comparison with previous results is shown in Table 10.

7 Conclusion

In this work, quantum circuits of AES are studied and optimized. We first propose an improved greedy algorithm based on de Brugière *et al.*'s greedy algorithm. When applied to many MDS matrices and matrices used in block ciphers, our improved greedy algorithm gives the best results with the lowest depth for all of them. For example, our improved method finds an in-place CNOT circuit of AES MixColumns with depth 10, which breaks the recent record of depth 16 and helps to reduce the full depth of AES. To further optimize quantum circuits of AES, we propose a new compressed pipeline structure for iterative building blocks whose round function can be taken as a unit. The Encryption oracle under the compressed pipeline structure will have the lowest round depth to date, and the encryption circuit under our structure will have better depth-width trade-off when the number of ancilla qubits of a round function is small

⁸ We contacted the author of [23] and learned that their not-yet-public circuit of Sbox with 68 ancilla qubits makes the combined Sbox and $SubS^\dagger$ require 93 ancilla qubits. Our circuit uses the input-invariant version of Sbox and has a maximum width of 3268 which is internally optimized via ProjectQ [56].

Table 10. Comparison of encryption circuit metrics from various sources

Source	#CNOT	#X	#Toffoli	$TofD$	W	$TofD-W$ cost
[20]	166,548	1,456	151,552	12,672	984	12,469,248
[3]	192,832	1,370	150,528	–	976	–
[35]	53,360	1,072	16,688	12,168	264	3,212,352
[30]	107,960	1,570	16,940	1,880	864	1,624,320
[63]	128,517	4,528	19,788	2,016	512	1,032,192
[22] ($p = 9$)	126,016	2,528	17,888	1,558	374	582,692
[35]	53,496	1,072	16,664	1,472	328	482,816
[22] ($p = 18$)	126,016	2,528	17,888	820	492	403,440
[23]	81,312	800	12,240	40	6,368	254,720
[36] ($m = 16$)	77,984	2,224	19,608	476	474	225,624
This paper^c	96,364	2,172	21,660	220	944	207,680
[38] (out-of-place)	75,024	800	12,920	40	4,823	192,920
[38] (in-place)	65,736	800	12,920	40	3,667	146,680
[23]	63,868	816	12,380	40	3,428	137,120
This paper^a	67,150	800	12,920	40	3,368	134,720
This paper^b	64,750	800	12,920	40	3,268	130,720

^a Using our improved shallowed pipeline structure and the input-invariant version of combined Sbox and Sbox[†] in [38].

^b Using our improved shallowed pipeline structure and the input-invariant version of combined Sbox and Sbox[†] with fewer qubits in [23].

^c Using our compressed pipeline structure and the C_1 circuit in [36] with $TofD$ 22 and 6 ancilla qubits.

enough. Detailed encryption circuits of AES under the guidance of our structure are given and compared with other circuits. Then an AES-128 Encryption oracle with T -depth 33 is synthesized. Finally, the shallowed pipeline structure is improved in two aspects of the key schedule and the Sbox, which leads to an encryption circuit with the lowest $TofD-W$ cost. Further optimization of the Sbox is left as a future work. Our methods in this paper can be used to optimize quantum circuits of other iterative building blocks.

Acknowledgements. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This research was supported by National Key Research and Development Project under Grant No. 2018YFA0704705 and CAS Project for Young Scientists in Basic Research (Grant No. YSBR-035).

A The Depth 10 Implementation of AES MixColumns

Table 11. The implementation of AES MixColumns with quantum depth 10. (i, j) stands for the CNOT gate adding the i -th qubit to the j -th qubit. The outputs $|y_0\rangle, |y_1\rangle, \dots, |y_{31}\rangle$ are represented by 0, 17, 18, 27, 28, 21, 22, 15, 16, 25, 26, 3, 20, 29, 30, 7, 24, 1, 10, 19, 12, 5, 6, 31, 8, 9, 2, 11, 4, 13, 14, 23, respectively.

Operation	Operation	Operation	Operation	Operation	Operation
Depth 1	(16, 0)	(22, 23)	(3, 23)	(1, 18)	(13, 21)
(12, 28)	(24, 8)	Depth 4	(8, 24)	(9, 26)	(12, 20)
(20, 4)	(26, 18)	(20, 27)	(17, 1)	(14, 23)	(24, 9)
(19, 3)	(15, 23)	(22, 31)	(31, 19)	(31, 8)	(19, 27)
(27, 11)	(2, 10)	(16, 17)	(22, 30)	(29, 13)	(8, 17)
(21, 5)	(29, 5)	(10, 18)	(7, 4)	(28, 12)	(26, 18)
(13, 29)	(3, 11)	(4, 21)	(2, 12)	(11, 4)	Depth 10
(6, 22)	(25, 17)	(25, 9)	(25, 18)	(6, 15)	(21, 5)
(30, 14)	(1, 9)	(7, 0)	(5, 21)	(20, 27)	(13, 29)
(23, 31)	(22, 14)	(29, 6)	Depth 7	(16, 25)	(30, 14)
(15, 7)	Depth 3	(2, 26)	(23, 24)	(0, 24)	(6, 22)
(18, 2)	(17, 26)	(11, 19)	(7, 18)	(10, 3)	(15, 7)
(26, 10)	(18, 19)	(3, 23)	(1, 2)	Depth 9	(23, 31)
(24, 1)	(20, 13)	(8, 24)	(16, 9)	(31, 7)	(27, 3)
(0, 8)	(31, 12)	Depth 5	(22, 19)	(29, 5)	(20, 4)
(9, 25)	(11, 3)	(18, 26)	(31, 20)	(2, 10)	(19, 11)
(16, 17)	(8, 9)	(27, 12)	(10, 27)	(22, 14)	(12, 28)
Depth 2	(21, 30)	(17, 9)	(5, 13)	(15, 23)	(9, 25)
(31, 7)	(28, 4)	(7, 3)	(28, 29)	(4, 28)	(26, 2)
(19, 27)	(2, 27)	(31, 15)	(25, 26)	(3, 11)	(18, 10)
(12, 20)	(7, 25)	(22, 8)	Depth 8	(25, 1)	(24, 16)
(13, 21)	(14, 6)	(23, 20)	(5, 22)	(30, 6)	(8, 0)
(4, 28)	(29, 15)	(24, 16)	(7, 17)	(0, 16)	(17, 1)
(30, 6)	(24, 16)	Depth 6			

B The Key Schedules and Cost Analysis of Our Compressed Pipeline Structure for AES-192 and AES-256

The key schedule of AES is based on 32-bit words. Denote the master key by W_0, W_1, \dots, W_{s-1} , where $s = 4, 6, 8$ for AES-128, AES-192 and AES-256, respec-

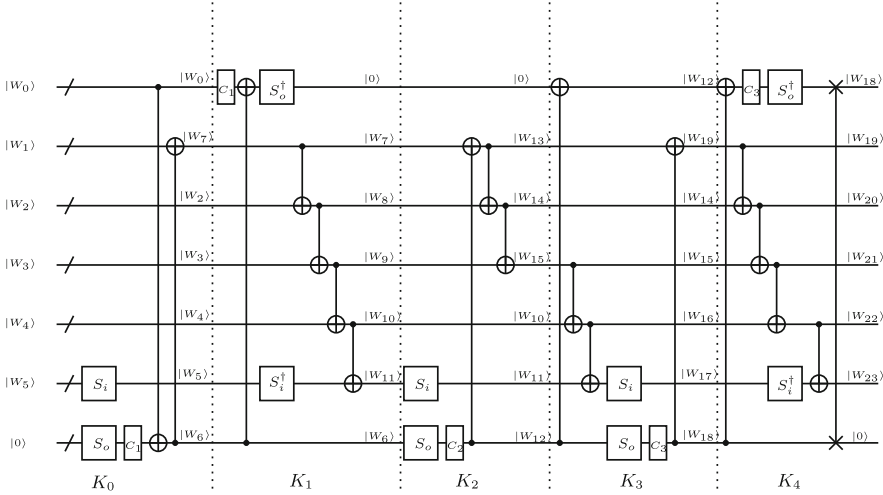


Fig. 14. The key schedule of our compressed pipeline structure for AES-192. S_i and S_o stands for the input and output registers of 4S-boxes, and C_j stands for $Const_j$.

tively. Except the given words (i.e., the words in the master key), 44, 52, 60 words are required by AES-128, AES-192 and AES-256, respectively. For all AES-128, AES-192 and AES-256, k_j^i equals W_{4j+i} and is the i -th 32-bit word of the j -th roundkey.

For AES-128, the word W_i can be calculated by the following equation:

$$W_i = \begin{cases} W_{i-4} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus \text{Const}(i/4), & \text{if } i \equiv 0 \pmod{4}, \\ W_{i-4} \oplus W_{i-1}, & \text{otherwise,} \end{cases}$$

where $i = 4, 5, \dots, 43$.

For AES-192, the word W_i can be calculated by the following equation:

$$W_i = \begin{cases} W_{i-6} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus \text{Const}(i/4), & \text{if } i \equiv 0 \pmod{6}, \\ W_{i-6} \oplus W_{i-1}, & \text{otherwise,} \end{cases}$$

where $i = 4, 5, \dots, 51$.

For AES-256, the word W_i can be calculated by the following equation:

$$W_i = \begin{cases} W_{i-8} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus \text{Const}(i/8), & \text{if } i \equiv 0 \pmod{8}, \\ W_{i-8} \oplus \text{SubWord}(W_{i-1}) & \text{if } i \equiv 4 \pmod{8}, \\ W_{i-8} \oplus W_{i-1}, & \text{otherwise,} \end{cases}$$

where $i = 4, 5, \dots, 59$.

Fewer additional key registers are needed for AES-192’s and AES-256’s key schedules compared to AES-128’s. For AES-192’s key schedule, only 1 additional 32-bit key register is needed. The circuits of the first 5 rounds are shown below

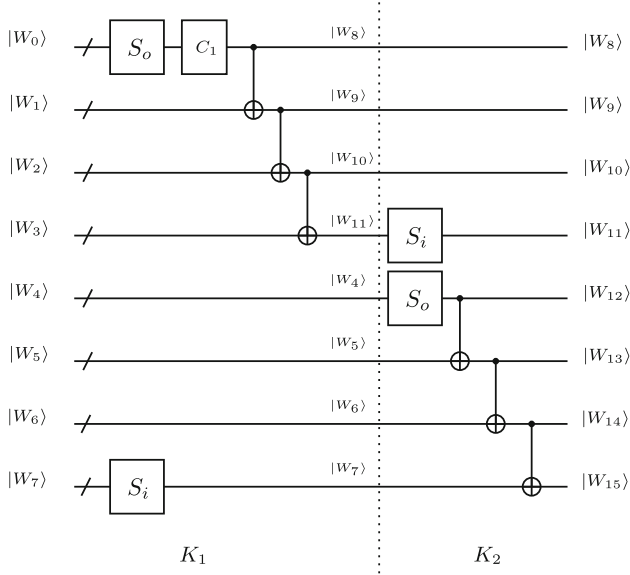


Fig. 15. The key schedule of our compressed pipeline structure for AES-256. S_i and S_o stands for the input and output registers of 4 S-boxes, and C_j stands for $Const_j$.

in Fig. 14, and the following rounds are designed similarly. In each round, there is one layer of 4 parallel S-boxes. Note that the 8 key registers needed after K_i are $W_{4i}, W_{4i+1}, W_{4i+2}, \dots, W_{4i+7}$. $W_{4i+2}, \dots, W_{4i+7}$ are computed in the circuit, and for W_{4i}, W_{4i+1} we have:

- $W_{4i+1} = W_{4i+6} \oplus W_{4i+7}$;
- $W_{4i} = W_{4i+5} \oplus W_{4i+6}$, if $4i \bmod 6 \neq 0$;
- W_{4i} is stored in the first key register and cleaned in the next round, if $4i \bmod 6 = 0$.

One can see that the additional key register is necessary due to the case of $4i \bmod 6 = 0$, if only one layer of S-boxes is allowed in each round.

For AES-256’s key schedule, no additional 32-bit key registers are needed. The circuits of K_1, K_2 are shown below in Fig. 15, and the following rounds are designed similarly. Note that the circuit of K_0 is an identity. In each round, there is one layer of 4 parallel S-boxes. After K_i the 8 key registers store two consecutive roundkeys exactly.

The resource costs of different structures for encryption circuits of AES-192 and AES-256 are given in Table 12 and Table 13, respectively. The bound m for better trade-offs of our AES-192 and AES-256 are more relaxed than AES-128 since the extra costs of key schedules are smaller. For AES-192, our circuit needs fewer ancilla qubits than C_p when $m < 70$, and has lower $ToFD-W$ cost than C_i for all m ; For AES-256, our circuit needs fewer ancilla qubits than C_p when $m < 88$, and has lower $ToFD-W$ cost than C_i for all m .

Table 12. Costs of encryption circuits of AES-192 for different structures

Circuits	C_p	C_i	our C_{cp}
Qubits of key registers	192	192	224
Qubits of message registers	128×13	128×2	128×4
C_1 circuits in parallel	16	16	32
C_2 circuits in parallel	4	2	4
Layers of AES S-box	12	24	12

Table 13. Costs of encryption circuits of AES-256 for different structures

Circuits	C_p	C_i	our C_{cp}
Qubits of key registers	256	256	256
Qubits of message registers	128×15	128×2	128×4
C_1 circuits in parallel	16	16	32
C_2 circuits in parallel	4	2	4
Layers of AES S-box	14	28	14

References

1. Aaronson, S., Gottesman, D.: Improved simulation of stabilizer circuits. *Physical Review A* **70**(5), 052328 (2004)
2. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçın, T.: Block ciphers—focus on the linear layer (feat. pride). In: *Advances in Cryptology—CRYPTO 2014: 34th Annual Cryptology Conference*, Santa Barbara, CA, USA, August 17–21, 2014, Proceedings, Part I 34. pp. 57–76. Springer (2014)
3. Almazrooie, M., Samsudin, A., Abdullah, R., Mutter, K.N.: Quantum reversible circuit of AES-128. *Quantum information processing* **17**, 1–30 (2018)
4. Amy, M., Maslov, D., Mosca, M.: Polynomial-time T-depth optimization of clifford+ T circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **33**(10), 1476–1489 (2014)
5. Amy, M., Maslov, D., Mosca, M., Roetteler, M.: A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **32**(6), 818–830 (2013)
6. Avanzi, R.: The QARMA block cipher family. Almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Transactions on Symmetric Cryptology* pp. 4–44 (2017)
7. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: *Advances in Cryptology—ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security*, Auckland, New Zealand, November 29–December 3, 2015, Proceedings, Part II 21. pp. 411–436. Springer (2015)
8. Banik, S., Funabiki, Y., Isobe, T.: Further results on efficient implementations of block cipher linear layers. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* **104**(1), 213–225 (2021)
9. Barreto, P., Nikov, V., Nikova, S., Rijmen, V., Tischhauser, E.: Whirlwind: a new cryptographic hash function. *Designs, codes and cryptography* **56**, 141–162 (2010)

10. Barreto, P.S.: The Anubis block cipher. NESSIE (2000)
11. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: *Advances in Cryptology—CRYPTO 2016: 36th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 14–18, 2016, Proceedings, Part II 36. pp. 123–153. Springer (2016)
12. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., et al.: Prince—a low-latency block cipher for pervasive computing applications. In: *Advances in Cryptology—ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security*, Beijing, China, December 2–6, 2012. Proceedings 18. pp. 208–225. Springer (2012)
13. de Brugière, T.G., Baboulin, M., Valiron, B., Martiel, S., Allouche, C.: Gaussian elimination versus greedy methods for the synthesis of linear reversible circuits. *ACM Transactions on Quantum Computing* **2**(3), 1–26 (2021)
14. Christof, B., Thorsten, K., Gregor, L.: Lightweight multiplication in $gf(2^n)$ with applications to mds matrices; crypto 2016. lncs 9814 (2016)
15. Cid, C., Murphy, S., Robshaw, M.J.: Small scale variants of the AES. In: *FSE*. vol. 3557, pp. 145–162. Springer (2005)
16. Daemen, J., Rijmen, V.: *The design of Rijndael*, vol. 2. Springer (2002)
17. Datta, K., Shrivastav, V., Sengupta, I., Rahaman, H.: Reversible logic implementation of AES algorithm. In: *2013 8th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. pp. 140–144. IEEE (2013)
18. De Brugière, T.G., Baboulin, M., Valiron, B., Martiel, S., Allouche, C.: Reducing the depth of linear reversible quantum circuits. *IEEE Transactions on Quantum Engineering* **2**, 1–22 (2021)
19. Fowler, A.G.: Time-optimal quantum computation. arXiv preprint [arXiv:1210.4626](https://arxiv.org/abs/1210.4626) (2012)
20. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying grover’s algorithm to AES: quantum resource estimates. In: *International Workshop on Post-Quantum Cryptography*. pp. 29–43. Springer (2016)
21. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. pp. 212–219 (1996)
22. Huang, Z., Sun, S.: Synthesizing quantum circuits of AES with lower T-depth and less qubits. In: *International Conference on the Theory and Application of Cryptology and Information Security*. pp. 614–644. Springer (2022)
23. Jang, K., Baksi, A., Kim, H., Song, G., Seo, H., Chattopadhyay, A.: Quantum analysis of AES. *Cryptology ePrint Archive* (2022)
24. Jaques, S., Naebrig, M., Roetteler, M., Virdia, F.: Implementing grover oracles for quantum key search on AES and LowMC. In: *Advances in Cryptology—EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30. pp. 280–310. Springer (2020)
25. Jean, J., Nikolić, I., Peyrin, T.: Joltik v1. 3. CAESAR Round **2** (2015)
26. Jean, J., Peyrin, T., Sim, S.M., Tourteaux, J.: Optimizing implementations of lightweight building blocks. *IACR Transactions on Symmetric Cryptology* **2017**(4), 130–168 (2017)
27. Jiang, J., Sun, X., Teng, S.H., Wu, B., Wu, K., Zhang, J.: Optimal space-depth trade-off of CNOT circuits in quantum logic synthesis. In: *Proceedings of the*

- Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 213–229. SIAM (2020)
28. Junod, P., Vaudenay, S.: FOX: a new family of block ciphers. In: Selected Areas in Cryptography: 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers 11. pp. 114–129. Springer (2005)
 29. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II 36. pp. 207–237. Springer (2016)
 30. Langenberg, B., Pham, H., Steinwandt, R.: Reducing the cost of implementing the advanced encryption standard as a quantum circuit. *IEEE Transactions on Quantum Engineering* **1**, 1–12 (2020)
 31. Leander, G., May, A.: Grover meets simon—quantumly attacking the FX-construction. In: Advances in Cryptology—ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23. pp. 161–178. Springer (2017)
 32. Li, S., Sun, S., Li, C., Wei, Z., Hu, L.: Constructing low-latency involutory MDS matrices with lightweight circuits. *IACR Transactions on Symmetric Cryptology* pp. 84–117 (2019)
 33. Li, Y., Wang, M.: On the construction of lightweight circulant involutory MDS matrices. In: Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. pp. 121–139. Springer (2016)
 34. Li, Z., Cai, B., Sun, H., Liu, H., Wan, L., Qin, S., Wen, Q., Gao, F.: Novel quantum circuit implementation of advanced encryption standard with low costs. *Science China Physics, Mechanics & Astronomy* **65**(9), 290311 (2022)
 35. Li, Z., Gao, F., Qin, S., Wen, Q.: New record in the number of qubits for a quantum implementation of AES. *Frontiers in Physics* **11**, 1171753 (2023)
 36. Lin, D., Xiang, Z., Xu, R., Zhang, S., Zeng, X.: Optimized quantum implementation of aes. *Quantum Information Processing* **22**(9), 352 (2023)
 37. Liu, M., Sim, S.M.: Lightweight MDS generalized circulant matrices. In: Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. pp. 101–120. Springer (2016)
 38. Liu, Q., Preneel, B., Zhao, Z., Wang, M.: Improved quantum circuits for AES: Reducing the depth and the number of qubits. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 67–98. Springer (2023)
 39. Liu, Q., Wang, W., Fan, Y., Wu, L., Sun, L., Wang, M.: Towards low-latency implementation of linear layers. *IACR Transactions on Symmetric Cryptology* pp. 158–182 (2022)
 40. Nielsen, M.A., Chuang, I.L.: *Quantum Computation and Quantum Information*. Cambridge University Press (2000)
 41. Patel, K.N., Markov, I.L., Hayes, J.P.: Optimal synthesis of linear reversible circuits. *Quantum Inf. Comput.* **8**(3), 282–294 (2008)
 42. Preskill, J.: Quantum computing in the NISQ era and beyond. *Quantum* **2**, 79 (2018)
 43. Q#, M.: Quantum development <https://devblogs.microsoft.com/qsharp/>
 44. Saeedi, M., Markov, I.L.: Synthesis and optimization of reversible circuits—a survey. *ACM Computing Surveys (CSUR)* **45**(2), 1–34 (2013)

45. Sarkar, S., Syed, H.: Lightweight diffusion layer: Importance of Toeplitz Matrices. *IACR Transactions on Symmetric Cryptology* **2016**(1), 95–113 (2016)
46. Sarkar, S., Syed, H.: Analysis of Toeplitz MDS Matrices. In: *Australasian Conference on Information Security and Privacy*. pp. 3–18. Springer (2017)
47. Schaeffer, B., Perkowski, M.: A cost minimization approach to synthesis of linear reversible circuits. arXiv preprint [arXiv:1407.0070](https://arxiv.org/abs/1407.0070) (2014)
48. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: Twofish: A 128-bit block cipher. *NIST AES Proposal* **15**(1), 23–91 (1998)
49. Selinger, P.: Quantum circuits of T-depth one. *Physical Review A* **87**(4), 042302 (2013)
50. Shende, V.V., Bullock, S.S., Markov, I.L.: Synthesis of quantum logic circuits. In: *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. pp. 272–275 (2005)
51. Shi, H., Feng, X., Xu, S.: A framework with improved heuristics to optimize low-latency implementations of linear layers. *IACR Transactions on Symmetric Cryptology* **2023**(4), 489–510 (2023)
52. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit block cipher CLEFIA. In: *Fast Software Encryption: 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26–28, 2007, Revised Selected Papers 14*. pp. 181–195. Springer (2007)
53. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: *Proceedings 35th annual symposium on foundations of computer science*. pp. 124–134. IEEE (1994)
54. Sim, S.M., Khoo, K., Oggier, F., Peyrin, T.: Lightweight MDS involution matrices. In: *Fast Software Encryption: 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8–11, 2015, Revised Selected Papers 22*. pp. 471–493. Springer (2015)
55. Simon, D.R.: On the power of quantum computation. *SIAM journal on computing* **26**(5), 1474–1483 (1997)
56. Steiger, D.S., Häner, T., Troyer, M.: Projectq: an open source software framework for quantum computing. *Quantum* **2**, 49 (2018)
57. Sun, X., Tian, G., Yang, S., Yuan, P., Zhang, S.: Asymptotically optimal circuit depth for quantum state preparation and general unitary synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023)
58. Wu, B., He, X., Yang, S., Shou, L., Tian, G., Zhang, J., Sun, X.: Optimization of CNOT circuits under topological constraints. arXiv preprint [arXiv:1910.14478](https://arxiv.org/abs/1910.14478) (2019)
59. Xiang, Z., Zeng, X., Lin, D., Bao, Z., Zhang, S.: Optimizing implementations of linear layers. *IACR Transactions on Symmetric Cryptology* pp. 120–145 (2020)
60. Zhang, A., Feng, X., Xu, S.: Size optimization of CNOT circuits on NISQ. arXiv preprint [arXiv:2210.05184](https://arxiv.org/abs/2210.05184) (2022)
61. Zhu, C., Huang, Z.: Optimizing the depth of quantum implementations of linear layers. In: *International Conference on Information Security and Cryptology*. pp. 129–147. Springer (2022)
62. Zou, J., Li, L., Wei, Z., Luo, Y., Liu, Q., Wu, W.: New quantum circuit implementations of SM4 and SM3. *Quantum Information Processing* **21**(5), 181 (2022)
63. Zou, J., Wei, Z., Sun, S., Liu, X., Wu, W.: Quantum circuit implementations of AES with fewer qubits. In: *Advances in Cryptology—ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*. pp. 697–726. Springer (2020)



Quantum Algorithms for Fast Correlation Attacks on LFSR-Based Stream Ciphers

Akinori Hosoyamada^{1,2(✉)}

¹ NTT Social Informatics Laboratories, Tokyo, Japan
akinori.hosoyamada@ntt.com

² NTT Research Center for Theoretical Quantum Information, Atsugi, Japan

Abstract. This paper presents quantum algorithms for fast correlation attacks, one of the most powerful techniques for cryptanalysis on LFSR-based stream ciphers in the classical setting. Typical fast correlation attacks recover a value related to the initial state of the underlying LFSR by solving a decoding problem on a binary linear code with the Fast Walsh-Hadamard Transform (FWHT). Applying the FWHT on a function in the classical setting is mathematically equivalent to applying the Hadamard transform on the corresponding state in quantum computation. While the classical FWHT on a function with ℓ -bit inputs requires $O(\ell 2^\ell)$ operations, the Hadamard transform on ℓ -qubit states requires only a parallel application of $O(\ell)$ basic gates. This difference leads to the exponential speed-up by some quantum algorithms, including Simon's period finding algorithm.

Given these facts, the question naturally arises of whether a quantum speedup can also be achieved for fast correlations by replacing the classical FWHT with the quantum Hadamard transform. We show quantum algorithms achieving speed-up in such a way, introducing a new attack model in the Q2 setting. The new model endows adversaries with a quite strong power, but we demonstrate its feasibility by showing that certain members of the ChaCha and Salsa20 families will likely be secure in the new model. Our attack exploits the link between LFSRs' state update and multiplication in a fine field to apply Shor's algorithm for the discrete logarithm problem. We apply our attacks on SNOW 2.0, SNOW 3G, and Sosemanuk, observing a large speed-up from classical attacks.

Keywords: Symmetric-key cryptography · Quantum cryptanalysis · Fast correlation attacks · LFSR-based stream ciphers

1 Introduction

While research and standardization of post-quantum public-key cryptosystems have been steadily progressing in the past decade [62], research on quantum security of symmetric-key cryptography has also been advancing. Starting with the early results of Grover's algorithm [39] for speeding up the exhaustive key search and the BHT algorithm [17] for speeding up collision search, a wide

variety of attack techniques have been proposed, including those breaking some schemes in polynomial time with Simon’s algorithm pioneered by Kuwakado and Morii [50, 51], Kaplan et al. [48], and Santoli and Scaffner [67]. A more recent research [14] has revealed that, even in the most conservative attack model, simply doubling the size of a secret key does not necessarily ensure the same level of security as in the classical setting. Another line of research started in [25, 44] has shown more rounds of some hash functions are broken in the quantum setting than in the classical setting, which underscores the importance of studying quantum attacks on symmetric key cryptography.

Whereas some quantum attacks are based on ideas completely different from classical attacks, others attempt to speed up classical attacks through quantum computation (e.g., [13, 49]). A large speed-up is sometimes obtained, while in other cases, an attack classically faster than the generic attack turns out to be slower than the quantum generic attack. To better understand security in the quantum setting, it is important to investigate how the efficiency and the validity of each classical attack change.

There are two attack models in the quantum setting, which are called Q1 and Q2 [49]. Q1 assumes that an adversary has a quantum computer, but oracles remain unchanged from the classical setting. In contrast, Q2 assumes both are quantum and that an oracle allows quantum superposition queries. The assumption of Q2 is strong, but Q2 attacks are still quite worth studying. If the key length of a target scheme is sufficiently long, Q2 attacks can be converted into Q1 by emulating the quantum oracle after getting all the outputs of the classical oracle (full codebook). Quite powerful Q1 attacks are sometimes developed from Q2 attacks [11, 14].

Quantum Fourier transforms (QFTs) play a crucial role in achieving exponential speedups in specific quantum algorithms, such as Shor’s [72] and Simon’s [74]. There are various types of QFTs, depending on the base group. For example, Shor’s algorithm utilizes QFT over the cyclic group of a large order. Meanwhile, Simon’s algorithm uses a QFT over $(\mathbb{Z}/2\mathbb{Z})^{\oplus n}$ for some n , which is referred to as the Hadamard transform. This transform is mathematically equivalent to the Walsh-Hadamard Transform (WHT) in classical computation.

The WHT has strong relationships with several classical attack techniques, particularly linear cryptanalysis [55]. Linear correlations of block ciphers can be obtained by applying the WHT, and the Fast-Walsh Hadamard Transform (FWHT) is commonly employed to accelerate key recovery [23]. It naturally raises the question of whether these traditional methods can be combined with the Hadamard transform to achieve quantum speedups. In fact, a recent work showed a framework to combine the quantum Hadamard transform and the classical linear key recovery attack with FHWT [69].

An important class of attacks closely linked with linear cryptanalysis is (fast) correlation attacks on LFSR-based stream ciphers. Correlation attacks, initially proposed by Siegenthaler [73], exploit linear correlations between keystreams output by a target cipher and the underlying LFSR’s output sequence. An enhanced version, today known as the fast correlation attack, was later given by Meier and Staffelbach [57]. Having been continually improved ever since [18, 20–22, 37, 46, 47, 58, 59, 75, 81, 83, 84], the fast correlation attack is currently the most

effective method for attacking various LFSR-based ciphers. For major ciphers such as SNOW 3G [31], research is being done to see how efficient the fast correlation attack can be, even if it is slower than the generic attack [35–37, 65, 80].

Roughly speaking, fast correlation attacks aim to recover the initial state of the underlying LFSR (or a related value) by solving the decoding problem of a linear code. Typical attacks perform the decoding quickly by applying the FWHT [22]. Given the aforementioned result on linear cryptanalysis in the quantum setting, the question naturally arises whether an interesting quantum attack can also be obtained for fast correlation attacks by replacing the classical FWHT with the Hadamard transform.

Based on the above motivation, this paper studies the quantum speedup of fast correlation attacks on LFSR-based stream ciphers. We focus on the setting where the decoding problem is defined over a binary code and a one-pass decoding algorithm with FWHT is applied, as this has been widely applied to various ciphers.

Technical Overview and Our Contributions. Before outlining our contributions, we briefly overview the basics of classical attacks.

Classical Fast Correlation Attacks on LFSR-Based Stream Ciphers. Typical LFSR-based stream ciphers are composed of an *initialization phase* and a *keystream generation phase*. The initialization phase takes a secret key K and an IV as input, non-linearly mixing them and loading the resulting values into internal registers. Following that, the keystream generation phase computes keystream bits, updating the internal states at each clock. Encryption and decryption are performed by XORing the keystream bits to a message or a ciphertext, as done in the counter mode. By the *initial state*, we denote the state right after the initialization phase.

As mentioned earlier, fast correlation attacks recover a value related to the LFSR's initial state by solving a decoding problem. In the simplest case, when there is a linear approximation with correlation c between the key stream output by the cipher and the output sequence of the underlying binary LFSR of length ℓ , a binary linear code is defined such that a message of ℓ bits (the initial state of the LFSR) is encoded to a codeword of length $N \gg \ell/c^2$. The basic idea is that if we regard the keystream as the encoded message with some noise added and decode it, correcting the errors, then we obtain the original message, namely the initial state of the LFSR.

Decoding is performed in the following manner. First, a certain function $\Psi(\mathbf{x})$ with ℓ -bit inputs (determined according to the binary code and keystream bits) is computed for all \mathbf{x} . Second, the WHT of the function Ψ , denoted by $\mathcal{W}(\Psi)$, is computed using FWHT with $O(\ell 2^\ell)$ operations. For each decoded message candidate \mathbf{x} , the larger the value $((\mathcal{W}(\Psi))(\mathbf{x}))^2$ is, the more likely it is that \mathbf{x} is the correct result. In particular, the decoding result is identified to be the \mathbf{x} that gives the maximum value of $((\mathcal{W}(\Psi))(\mathbf{x}))^2$. The computational complexity is $O(N + \ell 2^\ell)$ in total.

The decoding complexity $\ell 2^\ell$ is too large in most cases, making the attack slower than the exhaustive key search. To address the issue, a preprocessing procedure is usually performed in advance to reduce the dimension of the code (i.e., the parameter ℓ), thereby reducing the decoding complexity. However, the preprocessing procedure is usually heavy, and reducing the dimension increases the data complexity N . Hence, the preprocessing procedure and the number of dimensions to reduce are carefully adjusted to balance the computational complexity of preprocessing, the amount of data N , and the decoding complexity.

Next, we explain our results in the quantum setting.

Attempt in Q1. First, we try to obtain a quantum speed-up in the Q1 model by naturally extending classical attacks.

At the beginning of the attack, we obtain N bits of a keystream segment required to mount the attack (for some N). Next, we prepare a quantum state $|\psi\rangle$ corresponding to the function $\Psi(\mathbf{x})$. We show that the state $|\psi\rangle$ can be prepared with a complexity $\tilde{O}(N)$ in typical cases. Then, we apply the Hadamard transform to $|\psi\rangle$. The resulting quantum state $H^{\otimes \ell}|\psi\rangle$ is a quantum superposition of the message candidates $|\mathbf{x}\rangle$, among which the one with the largest quantum amplitude is the correct message. To find the correct \mathbf{x} , we apply the Quantum Amplitude Amplification (QAA) technique. The Boolean function required to apply QAA, denoted by f , can be chosen depending on the structure of the attack target. If the decoding problem is defined from a linear approximation with correlation c , the bit length of LFSR is ℓ , and the computational complexity to compute f is T_f , then the attack complexity becomes $O(N + 2^{\ell/2}T_f/\sqrt{Nc^2})$.

This is a quite natural extension of the classical attacks. However, we observe that applying the above algorithm to speed up existing classical attacks does not yield a quantum attack faster than the Grover search. Rather, we suspect it is quite hard to mount a fast correlation attack that is faster than the generic attack in the Q1 setting, or more fairly non-trivial techniques will be required. So, we focus on Q2 attacks.

Attack in Q2. An important feature of stream ciphers is that they can generate an exponentially long keystream from a single IV. In the Q2 setting, we first introduce a new attack model and security notion that reflects this feature well. Very roughly, the model allows an adversary to query the positions of keystream bits as well as IVs in quantum superposition. Although this attack model is very strong, we show that it is feasible in that some stream ciphers, such as some members of Chacha and Salsa20 families [8,9], are likely to achieve the security notion.

We then show a quantum decoding algorithm in the Q2 model. The underlying idea is the same as in Q1, but we make a non-trivial observation that the preparation of the quantum state $|\psi\rangle$ can be performed very efficiently using Shor's algorithm.

Recall that, in the Q1 attack, we first prepared the state $|\psi\rangle$ with both data and time complexity about N , which is exponentially large in usual scenarios. In

the Q2 setting, our strong attack model allows us to reduce the data complexity (i.e., the number of queries) from $O(N)$ to $O(1)$.

Our key observation is that even the time complexity can be reduced from $O(N)$ to polynomial time using Shor’s algorithm. In preparing the state $|\psi\rangle$, we need to solve the following problem: Given an arbitrary \mathbf{x} , find an index i such that \mathbf{x} equals to the i -th column of the generating matrix G of the code used in the attack. G is typically determined from the LFSR’s state update matrix and linear correlation masks. By leveraging the fact that the LFSR’s state update corresponds to the multiplication of an element generating $(\mathbb{F}_{2^\ell})^\times$, we find that the problem is reduced to a discrete logarithm problem in a typical case and can be efficiently solved with Shor’s algorithm.

As in the Q1 attack, QAA is applied to the state $H^{\otimes \ell}|\psi\rangle$ to amplify the quantum amplitude of the correct message. For all attack targets, we utilize the quantum counting algorithm to implement a Boolean function for QAA, similar to Kaplan et al.’s approach for quantum linear distinguishers [49]. As a result, the computational complexity of the attack is $O(\ell^4/c^2)$ when the attack is based on a code defined from a linear approximation with absolute correlation c . The value of c is large enough for some ciphers (around 2^{-20} in some cases) to achieve faster attacks than the exhaustive key recovery with Grover’s algorithm.

As applications, we show attacks on the ISO/IEC standard SNOW 2.0 [28, 45], SNOW 3G specified by 3GPP [31], and Sosemanuk in the eSTREAM portfolio [5, 26]. For SNOW 2.0, our attack works with time and query complexity $2^{59.3}$ and $2^{89.3}$, respectively. This is the first attack on the 256-bit key version of SNOW 2.0 faster than the Grover search¹. When our technique is applied to SNOW 3G, the resulting time and query complexity become $2^{102.9}$ and $2^{72.9}$, respectively. This is slower than the Grover search but significantly faster than classical attacks. (As mentioned earlier, research of attacks on SNOW 3G has actively been continued to determine how efficient fast correlation attacks can be [35–37, 65, 80], even though they are slower than the exhaustive key search.) About Sosemanuk, the time and query complexity of our attack becomes $2^{101.11}$ and $2^{73.15}$, respectively. This is slower than the quantum guess-and-determine attack in the Q1 model by Ding et al. [24], but faster than the Grover search when the key length is long (e.g., 256-bit). See also Table 1.

The quantum state $H^{\otimes \ell}|\psi\rangle$ appearing our attacks can be regarded as an analogy of the *correlation state* in the quantum linear key recovery attack by Schrottenloher [69], as the amplitude of each basis state $|\mathbf{x}\rangle$ in $H^{\otimes \ell}|\psi\rangle$ is, in fact, proportional to the correlation between a binary sequence derived from keystream and the codeword corresponding to \mathbf{x} . Still, the techniques used in the two attacks are quite different. The linear key recovery attack utilizes the Hadamard operator to compute some functions’ convolutions, performing (classical and) manual computation on the Walsh-Hadamard transform of a public function and exploiting a target cipher’s structure in which a subkey is XORed

¹ A previous work [24] showed an attack on SNOW 2.0 running in time about 2^{88} , but it requires exponentially many qubits (as large as 2^{88}) and in fact slower than the generic attack by the parallelized Grover search. More details are given in Remark 5.

Table 1. Comparison of attack complexity. The previous works define the time complexity unit as the time to perform arithmetic operations such as modular additions and finite field multiplications or, more ambiguously, as the time required to run the targeted cipher once. We regard the time complexity of an attack as the depth of the quantum circuit implementing it, where the depth is measured in T -gates and oracle gates.

Target	Key Length	Attack Model	Time	Data/Query	Ref./Note
SNOW 2.0	128/256	Classical Q2	$2^{162.86}$ $2^{89.3}$	$2^{159.62}$ $2^{59.3}$	[37] Sect. 6
SNOW 3G	128	Classical Q2	$2^{174.95}$ $2^{102.9}$	$2^{170.81}$ $2^{72.9}$	[35] Sect. 6
Sosemanuk	$0 \leq \kappa \leq 256$	Classical	$2^{134.8}$	2^{135}	[83]
		Q2	$2^{101.11}$	$2^{73.15}$	Sect. 6
		Q1	$\approx 2^{88}$	≈ 176	[24]
Any	κ	Q1/Q2	$\approx 2^{\kappa/2}$	$\approx \kappa$	Generic attack (Grover’s algorithm [39])

into a state. On the other hand, such convolution and manual computation of the Walsh-Hadamard transform do not appear in our attack, and Shor’s algorithm for discrete logarithms is utilized to prepare the state, exploiting the relationship between LFSRs’ state update and multiplication in a finite field.

Related Works. Independently and concurrently, Einsele and Semira also mentioned studies on quantum speed-up of fast correlation attacks [27], but only a short abstract is publicly available. In particular, concrete attack models or attack algorithms are not explained.

Paper Organization. Section 2 describes the notation, promises, and well-known basic results used in later chapters. Section 3 reviews classical fast correlation attacks. Section 4 discusses attacks in Q1. Section 5 introduces a new attack model and security notion, and Sect. 6 shows attacks in Q2.

2 Preliminaries

Unless otherwise noted, we assume all the vectors are row vectors. For any m and n , we naturally identify elements in \mathbb{F}_2^m with those in \mathbb{F}_2^{mn} , and mn -bit strings. For \mathbf{x} and \mathbf{y} in \mathbb{F}_2^m , $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbb{F}_2}$ denotes their formal inner product. The linear correlation between two binary sequences $\mathbf{x} = (x_0, \dots, x_{N-1})$ and $\mathbf{y} = (y_0, \dots, y_{N-1})$ is defined by

$$\text{Cor}(\mathbf{x}, \mathbf{y}) := \frac{\#\{i : x_i = y_i\} - \#\{i : x_i \neq y_i\}}{N}.$$

We identify Boolean functions $f : \{0, \dots, N - 1\} \rightarrow \mathbb{F}_2$ with binary sequences $(f(0), \dots, f(N - 1))$. The linear correlation between two Boolean functions $\text{Cor}(f, g)$ is naturally defined through this identification. The Walsh-Hadamard transform of a function $F : \mathbb{F}_2^n \rightarrow \mathbb{C}$, denoted by $\mathcal{W}(F)$, is the function from \mathbb{F}_2^n to \mathbb{C} defined as²

$$(\mathcal{W}(F))(z) = \frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle x, z \rangle_{\mathbb{F}_2}} F(x).$$

2.1 Quantum Computation

This paper assumes the readers are familiar with quantum computation (refer to, e.g., [64] for the basics). We adapt the quantum circuit model as a model for quantum computation, assuming arbitrary circuits composed of a finite number of Clifford+ T gates, quantum oracle gates (only in the Q2 model), and quantum Random Access Memory (qRAM) gates. Here, the quantum oracle gate of a function $f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ is the $(m + n)$ -qubit gate such that, given a quantum state of the form $\sum_{x,y} \alpha_{x,y} |x, y\rangle$ as an input, outputs the state $\sum_{x,y} \alpha_{x,y} |x, y \oplus f(x)\rangle$. About qRAM, this paper assumes a quantum-accessible *classical* memory is available to an adversary. Namely, for an arbitrarily created list of classical data (x_1, \dots, x_n) , the adversary is given quantum oracle access to the function $i \mapsto x_i$. CNOT gates are assumed to operate on an arbitrary pair of qubits in a circuit. Quantum error correction is assumed to be perfectly performed with its cost being ignored. All the measurements are performed in the computational basis.

How to Evaluate Attack Costs. We always measure the depth of quantum circuits by calculating the depth in T gates, oracle gates, and qRAM gates. We regard the running time of a quantum circuit as its depth in this measure. When considering an attack on an LFSR-based stream cipher, we assume that the number of qubits available for an adversary is in a small polynomial of the underlying LFSR’s bit length to enable a fair comparison with the generic key-recovery attack using Grover’s algorithm [39] without parallelization.

Quantum Amplitude Amplification. Let U be a unitary operator acting on n -qubit quantum states, $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ be a Boolean function, and p be the probability that an x satisfying $f(x) = 1$ is obtained when the quantum state $U|0^n\rangle$ is measured. The Quantum Amplitude Amplification (QAA) technique [16] amplifies the probability p by making $O(p^{-1/2})$ quantum queries to f with $O(p^{-1/2})$ applications of U and U^\dagger as follows.

Proposition 1 (Plain QAA). *Let \mathcal{S}_f (resp., \mathcal{S}_0) be the unitary operators that multiplies the basis state $|x\rangle$ by $(-1)^{f(x)}$ (by (-1) iff $x = 0^n$), and define a unitary operator $Q(U, f) := -U\mathcal{S}_0U^\dagger\mathcal{S}_f$. When the quantum state $(Q(U, V))^i U|0^n\rangle$*

² We adopt the definition with the coefficient $1/\sqrt{2^n}$ to make it consistent with the Hadamard operators in the quantum setting.

is measured, an x satisfying $f(x) = 1$ is obtained with probability $\sin^2((2i + 1) \arcsin(\sqrt{p}))$, which is at least $\max(1 - p, p)$ when $i := \lfloor \frac{\pi}{4 \arcsin(\sqrt{p})} \rfloor$.

QAA with Certainty. If an adversary knows the exact value of p , then the QAA can be modified in such a way to obtain a good state with certainty, by modifying U to slightly lower the probability p to make $(\pi/(4 \arcsin(\sqrt{p})) - (1/2))$ be an integer [16]. In this paper, we assume the cost of this modification is negligible compared to implementing U and U^\dagger themselves, and QAA returns an x satisfying $f(x) = 1$ by applying U , U^\dagger , and \mathcal{S}_f at most $\arcsin(\sqrt{p}) \leq p^{-1/2}$ times (if an adversary knows the exact value of p).

QAA Without Knowing p . When applying the plain QAA in Proposition 1, the success probability does not become large enough not only if i is too small but also if i is too large. For instance, if $i \approx 2 \cdot \lfloor \frac{\pi}{4 \arcsin(\sqrt{p})} \rfloor$, then the success probability may be as small as p .

However, it is not necessarily easy to find the exact value of p , when it is practically too hard to compute the value $\lfloor \frac{\pi}{4 \arcsin(\sqrt{p})} \rfloor$ exactly. Even in such a case, an x satisfying $f(x) = 1$ can be found by running the plain QAA multiple times with random i as follows [15,16].

Algorithm QAAw/oKp.

1. Let $\alpha := 1$ and $\lambda := 6/5$.
2. Choose i from $\{0, 1, \dots, \alpha - 1\}$ uniformly at random.
3. Run the plain QAA with i iterations and measure the entire state, i.e., $(Q(U, f))^i U |0^n\rangle$. Let x be the measurement result.
4. If $f(x) = 1$, return x as the output. Otherwise, set $\alpha := \min \{ \lambda \cdot \alpha, \sqrt{2^n} \}$ and go to Step 2.

Proposition 2 (QAA without knowing p [15,16]). *Suppose $p \leq 3/4$. Then, the algorithm QAAw/oKp returns x satisfying $f(x) = 1$ with an expected number of applications of $Q(U, f)$ at most $(9/2)p^{-1/2}$.*

Grover’s algorithm [39] is the special case of QAA when $U = H^{\otimes n}$.

Quantum Counting Algorithm. Let QFT_q denote the quantum Fourier transform over $\mathbb{Z}/q\mathbb{Z}$. For any unitary operator W acting on n -qubit states and any positive integer q that is a power of 2, let $\Lambda_q(W)$ be the operator acting on $(\log q + n)$ -qubit states such that $\Lambda_q(W)|i\rangle|x\rangle = |i\rangle(W^i|x\rangle)$. Here, $0 \leq i \leq q - 1$ and $x \in \mathbb{F}_2^n$. For a unitary operator U and a Boolean function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, define the probability p and the operator $Q(U, f)$ as in Proposition 1. In addition, let $\text{Calc}_{n,q}$ be the unitary operator that, given a (classical) value θ , computes $2^n \cdot \sin^2(\pi\theta/q)$ and write the result into an additional register. Now, consider running the following algorithm without measurement.

Algorithm QC. Prepare $|0^{\log_2 q}\rangle|0^n\rangle$ as the initial state. Apply $(QFT_q \otimes H^{\otimes n})$, $\Lambda_q(Q(H^{\otimes n}, f))$, and then $(QFT_q^\dagger \otimes I_n)$ in sequential order. Finally, apply $\text{Calc}_{n,q}$,

taking input from the left log q -bit register and writing the output into an auxiliary register.

Proposition 3 ([16]). *Let $Z := |f^{-1}(1)|$. If the above algorithm QC is run and the auxiliary register is measured, then a value \tilde{Z} satisfying*

$$|Z - \tilde{Z}| \leq 2\pi\sqrt{Z(2^n - Z)}/q + (2^n \cdot \pi^2)/q^2 \tag{1}$$

is obtained with probability at least 0.8.

The depth to implement QC is typically dominated by that of $A_q(Q(H^{\otimes n}, f))$, which makes exactly q queries to f . We can show that $A_q(Q(H^{\otimes n}, f))$ can be implemented on a quantum circuit of depth at most about $q \cdot D_f$ by using n auxiliary qubits, where D_f is the depth to implement the quantum oracle of f . Hence, the depth of QC is also at most about $q \cdot D_f$, and the amount of the auxiliary qubits needed is at most the number of qubits required to compute $\text{Calq}_{n,q}$. See Section B of the full version of this paper [42] for more details.

2.2 LFSR Basics

Let \mathbb{F}_q be a finite field of order q . The LFSR on \mathbb{F}_q of length L with a feedback polynomial $f(x) := c_Lx^L + c_{L-1}x^{L-1} + \dots + c_1x + 1 \in \mathbb{F}_q[x]$ generates an infinite sequence $(s_t)_{t \geq 0}$ in \mathbb{F}_q from an initial state $\mathbf{s}^{(0)} = (s_0, \dots, s_{L-1}) \in \mathbb{F}_q^L$ as

$$s_{t+L} := \sum_{1 \leq i \leq L} c_i s_{t+L-i} \text{ for } t \geq 0,$$

maintaining the internal state $\mathbf{s}^{(t)} := (s_t, \dots, s_{t+L-1})$ at time t . The state update can be regarded as a linear map over \mathbb{F}_q , and $\mathbf{s}^{(t+1)} = \mathbf{s}^{(t)} \cdot M$ holds for

$$M := \begin{pmatrix} 0 & 0 & \dots & 0 & c_L \\ 1 & 0 & \dots & 0 & c_{L-1} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & c_1 \end{pmatrix}. \tag{2}$$

A well-known fact is that the period of LFSR sequences and internal states becomes the longest (i.e., $q^L - 1$) when f is a primitive polynomial.

Throughout the paper, we only consider LFSRs whose feedback polynomial f is primitive and assume that q is a power of 2.

The reciprocal polynomial of f is called the *characteristic polynomial* of the LFSR, which we denote by $f^*(x)$ (that is, $f^*(x) = x^L f(1/x)$). As we assume that f is primitive (and thus irreducible), so is f^* . Hence, the quotient ring $\mathfrak{F} := \mathbb{F}_q[x]/(f^*(x))$ becomes a field, which is isomorphic to \mathbb{F}_q^L as vector spaces over \mathbb{F}_q . Let $\xi : \mathbb{F}_q^L \rightarrow \mathfrak{F}$ be the isomorphism defined by

$$\xi(\mathbf{a}) = \sum_{0 \leq i \leq L-1} a_i \cdot \alpha^i \tag{3}$$

for $\mathbf{a} = (a_0, \dots, a_{L-1}) \in \mathbb{F}_q^L$, where $\alpha := x + (f^*(x)) \in \mathfrak{F}$ is a generator element of \mathfrak{F} over \mathbb{F}_q . Since q is assumed to be a power of 2, some straightforward calculations show

$$\xi(\mathbf{a} \cdot M^\top) = \xi(\mathbf{a}) \cdot \alpha. \tag{4}$$

Since f^* is not only irreducible but also primitive, α is a generator of the multiplicative group $\mathfrak{F}^\times \cong \mathbb{Z}/(q^L - 1)\mathbb{Z}$, and so $\beta \cdot \alpha^i \neq \beta$ holds for arbitrary $\beta \in \mathfrak{F} \setminus \{0\}$ and $i = 1, \dots, q^L - 2$. From this fact and Eq. (4),

$$\mathbf{a} \cdot (M^\top)^i \neq \mathbf{a} \text{ for } i = 1, \dots, q^L - 2 \tag{5}$$

follows for $\mathbf{a} \in \mathbb{F}_q^L \setminus \{\mathbf{0}\}$.

3 Classical Fast Correlation Attack

This section briefly reviews classical fast correlation attacks related to our results. We focus on so-called one-pass algorithms working using FWHT [22] that can be regarded as a decoding procedure for a binary linear code, as it has been most widely applied. First, we explain the simplest case where LFSR sequences themselves are correlated with keystreams in Sect. 3.1. Then, Sect. 3.2 explains how the attack idea is extended to more general cases. Section 3.3 gives a brief summary and a note on the amount of necessary data. Throughout the section, we assume an adversary is given a keystream segment produced from a single pair of a key and an IV. See, e.g., [2, 19, 56], for more details on classical fast correlation attacks.

3.1 Simplest Case

Suppose a stream cipher is built upon a single LFSR of length L over \mathbb{F}_2 and we have an N -bit keystream segment $\mathbf{z} = (z_0, z_1, \dots, z_{N-1}) \in \mathbb{F}_2^N$. Our goal is to recover the initial state $\mathbf{s}^{(0)} \in \mathbb{F}_2^L$ of the LFSR. Once $\mathbf{s}^{(0)}$ is recovered, it is often easy to determine the entire initial state, and even the master secret key is recovered in some cases.

In the simplest case, the fast correlation attack models that the keystream \mathbf{z} is obtained by transmitting the LFSR sequence $\mathbf{s} = (s_0, \dots, s_{N-1}) \in \mathbb{F}_2^N$ generated from $\mathbf{s}^{(0)}$ through a Binary Symmetric Channel (BSC). Namely, it regards as if $e_i := z_i \oplus s_i$ were an independent random error bit for each i (see Fig. 1), expecting that the error bit sequence $\mathbf{e} = (e_0, \dots, e_{N-1})$ is highly biased. Note that \mathbf{e} is biased iff the squared linear correlation $\text{Cor}(\mathbf{s}, \mathbf{z})^2$ is large. For ease of explanation, we assume e_i is biased to 0 and (the expected value of) the correlation $c := \mathbf{E}\mathbf{x}_{K,IV}[\text{Cor}(\mathbf{s}, \mathbf{z})]$ is close to 1.

In this model, the problem of recovering $\mathbf{s}^{(0)}$ from \mathbf{z} can be regarded as a decoding problem with respect to a binary linear code. Let G be the binary $L \times N$ matrix of which the i -th column vector is $M^{i-1} \cdot (1, 0, \dots, 0)^\top$. Then $\mathbf{s} = \mathbf{s}^{(0)}G$ holds by definition of LFSR. (Multiplication by M corresponds to clocking the LFSR once, and so multiplying by G generates the sequence $\mathbf{s} = (s_0, \dots, s_{N-1})$.) In addition, G is full-rank if N is sufficiently large. Especially, G can be regarded

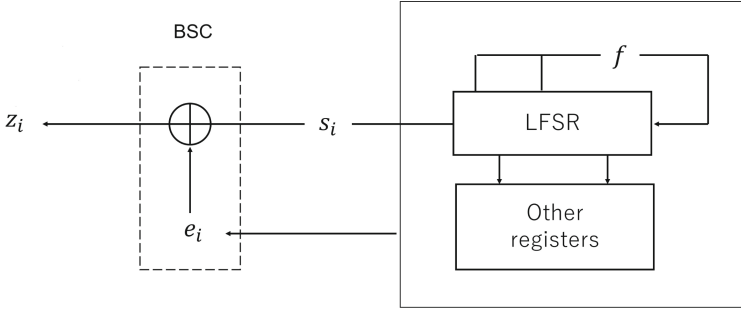


Fig. 1. LFSR-based cipher modeled as a BSC.

as a generating matrix of an $[N, L]$ binary linear code \mathcal{C} , where encoding a message vector corresponds to multiplying G from right. The initial state $\mathbf{s}^{(0)}$ corresponds to an original message before the encoding, and the LFSR sequence $\mathbf{s} = \mathbf{s}^{(0)}G$ to the codeword of \mathcal{C} after the encoding. From this perspective, recovering $\mathbf{s}^{(0)}$ from $\mathbf{z} = \mathbf{s} \oplus \mathbf{e}$ is equivalent to correcting errors and recovering the original message.

Concretely, $\mathbf{s}^{(0)}$ is recovered by maximum likelihood decoding, which can be realized roughly as follows.

1. For each candidate message $\mathbf{x} \in \mathbb{F}_2^L$, compute and store the squared linear correlation $\text{Cor}(\mathbf{x}G, \mathbf{z})^2$ between the codeword $\mathbf{x}G \in \mathcal{C} (\subset \mathbb{F}_2^N)$ and \mathbf{z} .
2. If $\mathbf{x} = \mathbf{s}^{(0)}$, the value $\text{Cor}(\mathbf{x}G, \mathbf{z})^2 = \text{Cor}(\mathbf{s}, \mathbf{z})^2$ will be large by assumption. On the other hand, it will be small for a random $\mathbf{x} \neq \mathbf{s}^{(0)}$. With this in mind, output \mathbf{x} with the largest $\text{Cor}(\mathbf{x}G, \mathbf{z})^2$ as the decoding result.

For each \mathbf{x} , computing $\text{Cor}(\mathbf{x}G, \mathbf{z})^2$ requires $O(N)$ operations because we have to check whether $(\mathbf{x}G)_i = z_i$ for $i = 0, \dots, N - 1$. Hence this procedure requires $O(2^L \cdot N)$ operations in total. To achieve a high success probability, $N \geq \Omega(L/c^2)$ is necessary due to Shannon’s noisy-channel coding theorem, and some statistical analysis shows that $N = O(L/c^2)$ is indeed sufficient (we will elaborate this later in Sect. 3.3).

By applying the Fast Walsh-Hadamard Transform (FWHT), the decoding complexity drops from $O(N \cdot 2^L)$ to $O(N + L2^L)$. Define a function $\Psi : \mathbb{F}_2^L \rightarrow \mathbb{C}$ by

$$\Psi(\mathbf{w}) := \sum_{\substack{0 \leq i \leq N-1: \\ \mathbf{w} = \text{the } (i+1)\text{-th column of } G}} (-1)^{z_i}. \tag{6}$$

Compute and store $\Psi(\mathbf{w})$ for all \mathbf{w} , which can be done with $O(N)$ operations and $O(2^L)$ memory. Then, apply the FWHT to compute and store the value $(\mathcal{W}(\Psi))(\mathbf{x})$ for all \mathbf{x} , which requires $O(L2^L)$ operations. Now, some straightforward calculations³ show

³ By definition of Ψ , \mathcal{W} , and Cor , it immediately follows that both sides are equal to $\frac{1}{2^{L/2}} \sum_{\mathbf{w}, i} (-1)^{(\mathbf{x}, \mathbf{g}_i)_{\mathbb{F}_2} \oplus z_i} \delta_{\mathbf{w}, \mathbf{g}_i}$, where \mathbf{g}_i is the $(i + 1)$ -th column of G .

$$(\mathcal{W}(\Psi))(\mathbf{x}) = \frac{N}{2^{L/2}} \cdot \text{Cor}(\mathbf{x}G, \mathbf{z}).$$

Hence, the first step of the aforementioned decoding procedure can be performed with $O(N + L2^L)$ operations.

3.2 More General Cases

Modern stream ciphers are well-designed so that keystreams themselves are not strongly correlated with LFSR sequences, and the above attack does not work. Yet, almost the same idea is applicable if there is another code relating initial states and keystreams.

For instance, suppose

- there are (1) the generating matrix G of an $[N, \ell]$ binary code for some ℓ , (2) a binary sequence $\zeta := (\zeta_0, \dots, \zeta_{N-1})$ computed from a keystream segment, and (3) a value $\sigma^{(0)} \in \mathbb{F}_2^\ell$ that is related to the initial value $\mathbf{s}^{(0)}$, such that
- (the absolute value of the expected value of) the correlation $c := |\mathbf{E}\mathbf{x}_{K,IV} [\text{Cor}(\sigma^{(0)}G, \zeta)]|$ is large.

Then, the aforementioned decoding algorithm with FWHT works in exactly the same way, except that now the decoding algorithm recovers $\sigma^{(0)}$ and the parameters and variables such as L and z_i are replaced with ℓ and ζ_i , etc. The decoding complexity with FWHT becomes $O(N + \ell 2^\ell)$, and $N \geq \Omega(\ell/c^2)$ is required for a sufficiently high success probability. Once $\sigma^{(0)}$ is recovered, at least the keystream is distinguished from random, and sometimes it is possible to recover the entire initial state and even the master secret key of the cipher.

A typical way to find such an alternative code is to search for a linear approximation between internal states of an LFSR and keystreams. Suppose that an attack target is based on an LFSR of length L over \mathbb{F}_{2^n} for some n , and that the LFSR sequence (resp., keystream) is denoted by $s_0, s_1, \dots \in \mathbb{F}_{2^n}$ (resp., $z_0, z_1, \dots \in \mathbb{F}_{2^n}$). As before, let $\mathbf{s}^{(i)} := (s_i, \dots, s_{i+L-1}) \in \mathbb{F}_{2^n}^L$ be the internal state of the LFSR at time i . Assume there are an index set $\mathbf{I}_{\text{lfsr}} \subset \mathbb{Z}_{\geq 0}$ and linear masks $\{\Gamma_j\}_{j \in \mathbf{I}_{\text{lfsr}}} \subset \mathbb{F}_{2^n}^L$ for the LFSR's internal states (resp., an index set $\mathbf{I}_{\text{ks}} \subset \mathbb{Z}_{\geq 0}$ and linear masks $\{\mathbf{A}_j\}_{j \in \mathbf{I}_{\text{ks}}} \subset \mathbb{F}_{2^n}$ for keystreams) such that the linear approximation

$$\bigoplus_{j \in \mathbf{I}_{\text{lfsr}}} \langle \mathbf{s}^{(i+j)}, \Gamma_j \rangle_{\mathbb{F}_2} \approx \bigoplus_{j \in \mathbf{I}_{\text{ks}}} \langle z_{i+j}, \mathbf{A}_j \rangle_{\mathbb{F}_2} \tag{7}$$

holds with an absolute correlation $c \gg 0$ for every i . Below, we explain how to define G , ζ , and $\sigma^{(0)}$ such that $|\mathbf{E}\mathbf{x}_{K,IV} [\text{Cor}(\sigma^{(0)}G, \zeta)]| \approx c$ from the above linear approximation.

Define $\mathbf{I} \in \mathbb{F}_2^{L_n}$ by $\mathbf{I} := \bigoplus_{j \in \mathbf{I}_{\text{fsr}}} \left(\mathbf{I}_j \cdot (M^\top)^j \right)$. Then we have

$$\langle \mathbf{s}^{(0)}, \mathbf{I} \cdot (M^\top)^i \rangle_{\mathbb{F}_2} = (\text{the left hand side of (7)}).$$

With this in mind, setting $\ell := L \cdot n$ (and identifying $\mathbb{F}_2^{L_n}$ with \mathbb{F}_2^ℓ), define

- G as the $\ell \times N$ binary matrix of which the i -th column is $M^{i-1} \cdot \mathbf{I}^\top$,
- $\boldsymbol{\sigma}^{(0)} := \mathbf{s}^{(0)}$, and
- $\boldsymbol{\zeta} = (\zeta_0, \dots, \zeta_{N-1})$ by $\zeta_i := (\text{the right hand side of (7)})$.

Then, Eq. (7) can be rewritten as

$$(\boldsymbol{\sigma}^{(0)} G)_i \approx \zeta_i,$$

which implies $|\mathbf{E}_{\mathbf{x}_{K,IV}} [\text{Cor}(\boldsymbol{\sigma}^{(0)} G, \boldsymbol{\zeta})]| \approx c$.

Usual attacks further convert the above G into another matrix G' to reduce the code's dimension and the decoding complexity, at the cost of a decrease in the (squared) correlation and an increase in the data complexity. This is usually done by solving some k -sum problems with Wagner's k -tree algorithm [77].

3.3 Summary and Note on the Size of N

To mount fast correlation attacks, an attacker first looks for an $\ell \times N$ matrix G , along with a binary sequence $\boldsymbol{\zeta} = (\zeta_0, \dots, \zeta_{N-1})$ that can be computed from a keystream segment, such that $c := |\mathbf{E}_{\mathbf{x}_{K,IV}} [\text{Cor}(\boldsymbol{\sigma}^{(0)} G, \boldsymbol{\zeta})]|$ is large for some $\boldsymbol{\sigma}^{(0)} \in \mathbb{F}_2^\ell$ that depends on the (secret) initial value $\mathbf{s}^{(0)}$ of the LFSR. Here, G is public and the adversary can compute it offline.

Once finding such G , $\boldsymbol{\zeta}$, and $\boldsymbol{\sigma}^{(0)}$, the adversary performs maximum likelihood decoding of $\boldsymbol{\zeta}$ with respect to the $[N, \ell]$ binary linear code of which the generating matrix is G . The decoding can be realized with $O(N + \ell 2^\ell)$ operations as follows.

1. Compute all the values of the function $\Psi(\mathbf{z}) := \sum_{0 \leq i \leq N-1: \mathbf{z} = \text{the } (i+1)\text{-th column of } G} (-1)^{\zeta_i}$ and store them into a memory.
2. Apply the FWHT on $\Psi(\mathbf{z})$. Now, the values $(\mathcal{W}(\Psi))(\mathbf{x}) = \frac{N}{2^{i/2}} \cdot \text{Cor}(\mathbf{x}G, \boldsymbol{\zeta})$ are stored in the memory for all \mathbf{x} .
3. Output \mathbf{x} such that $\text{Cor}(\mathbf{x}G, \boldsymbol{\zeta})^2$ is significantly larger than others.

G and $\boldsymbol{\zeta}$ are typically derived from linear approximations between LFSR sequences and keystreams.

Very roughly and intuitively, $\boldsymbol{\sigma}^{(0)}$ corresponds to (a linear transformation of) the initial state of LFSR, $\boldsymbol{\sigma}^{(0)} G$ to the output sequence of LFSR, and $\boldsymbol{\zeta}$ to the key stream. If $\boldsymbol{\zeta}$ is linearly approximated by $\boldsymbol{\sigma}^{(0)} G$, then $\boldsymbol{\zeta}$ can be regarded as the result of encoding $\boldsymbol{\sigma}^{(0)}$ with a code corresponding to G and sending through a noisy channel. Hence, $\boldsymbol{\sigma}^{(0)}$ (and thus the initial state of the LFSR) can be recovered by the most likelihood decoding, using FWHT as above.

About the Size of N . Here we explain why $N = O(\ell/c^2)$ is sufficient to achieve a large success probability. Let us call $\sigma^{(0)}$ the correct decoding results, and $\mathbf{x} \in \mathbb{F}_2^\ell$ such that $\mathbf{x} \neq \sigma^{(0)}$ incorrect decoding results. We heuristically assume that, for an incorrect \mathbf{x} , the correlation $\text{Cor}(\mathbf{x}G, \zeta)$ is approximated by the linear correlation of two random binary sequences of length N , as done in classical attacks. Then the following claim holds.

Claim Suppose $N \geq 8\ell/c^2$ and $\ell \geq 1$. Then we have

$$\Pr_{K,IV} \left[\text{There is } \mathbf{x} \neq \sigma^{(0)} \text{ such that } \text{Cor}(\mathbf{x}G, \zeta)^2 \geq c^2/4 \right] \lesssim (2/e)^\ell, \quad (8)$$

$$\Pr_{K,IV} \left[\text{Cor}(\sigma^{(0)}G, \zeta)^2 \geq c^2/2 \right] \gtrsim 0.95. \quad (9)$$

Especially, the decoding algorithm succeeds with a sufficiently high probability. See Section C of the full version of this paper [42] for why it is plausible to regard that this claim holds.

4 Quantum Fast Correlation Attack in the Q1 Model

This section studies quantum speed-up of the decoding procedure of fast correlation attacks with the FWHT in the Q1 model. In fact, it later turns out that it seems hard to achieve a fast correlation attack that is faster than the Grover search in the Q1 model by speeding-up existing classical attacks. Yet, we show a Q1 algorithm here to make it the starting point of a more complex Q2 attack in Sect. 6, and to see why achieving a meaningful speed-up of existing classical fast correlation attacks seems hard in Q1.

As in the classical setting, we assume that a keystream segment generated from a single key and IV pair is given to an adversary. We consider the general cases reviewed in Sect. 3.2, and use the same notations.

Below, we first describe a rough idea of the quantum attack in Sect. 4.1, and then provide the formal details in Sect. 4.2. Section 4.3 provides discussions on applications and some observations.

4.1 Overview and Rough Idea

Our idea is to perform quantum analogue of the operations in the classical decoding procedure in a natural way.

- We first prepare the quantum counter part of the function Ψ , namely the quantum state

$$|\psi\rangle := \sum_{\mathbf{w} \in \mathbb{F}_2^\ell} \frac{\Psi(\mathbf{w})}{\sqrt{\sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2}} |\mathbf{w}\rangle. \quad (10)$$

How we can prepare $|\psi\rangle$ is a non-trivial question, but we show that a unitary operator U satisfying $U|0^\ell\rangle = |\psi\rangle$ can be realized as an efficient quantum algorithm, given that some data are precomputed and stored in qRAM in advance.

- Second, we apply the Hadamard transform on the entire state. Since the Walsh-Hadamard transform on classical functions is mathematically the same as the Hadamard transform on quantum states, we get

$$H^{\otimes \ell}|\psi\rangle = \sum_{\mathbf{x} \in \mathbb{F}_2^\ell} \frac{(\mathcal{W}(\Psi))(\mathbf{x})}{\sqrt{\sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2}} |\mathbf{x}\rangle = \sum_{\mathbf{x} \in \mathbb{F}_2^\ell} \frac{N \cdot \text{Cor}(\mathbf{x}G, \zeta)}{\sqrt{\sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2 \cdot 2^{\ell/2}}} |\mathbf{x}\rangle. \quad (11)$$

Measuring this state, we obtain an \mathbf{x} with a probability proportional to the squared correlation $\text{Cor}(\mathbf{x}G, \zeta)^2$. Namely, we will obtain the correct decoding result $\sigma^{(0)}$ with a higher probability than incorrect results. However, the probability to obtain $\sigma^{(0)}$ is usually still too small.

- Thus, we amplify the probability of obtaining a correct result with QAA. To apply QAA, we must implement a unitary operator computing the Boolean function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ such that $f(\mathbf{x}) = 1$ iff $\mathbf{x} = \sigma^{(0)}$. How to choose and implement f can depend on the internal structure of the target cipher.

4.2 Formal Details

First, we explain some precomputation required for later steps. Second, we show how to prepare the state $|\psi\rangle$ in Eq. (10). Third, we provide a formal description and analysis of the entire attack algorithm.

We denote the i -th column vector of G by \mathbf{g}_i , and define $\mu := \max_{\mathbf{x} \in \mathbb{F}_2^\ell} \#\{i : \mathbf{g}_i = \mathbf{x}\}$. We suppose that $|\mathbf{E}\mathbf{x}_{K,IV} [\text{Cor}(\sigma^{(0)}G, \zeta)]| = c$ for some $c \gg 0$ and $8\ell/c^2 \leq N \leq 2^\ell$.

Precomputation. Given sufficient amount of keystream bits, we first compute $\zeta = (\zeta_0, \dots, \zeta_{N-1})$ and store them into qRAM. Then, we compute \mathbf{g}_i and store the data (i, \mathbf{g}_i) into a list in a sequential order for all i . Along with \mathbf{g}_i , store the information of how many times the value \mathbf{g}_i has appeared before. That is, store a counter set to be 0 if $\mathbf{g}_j \neq \mathbf{g}_i$ for all $j < i$, and increment it to 1 if there is unique $j < i$ such that $\mathbf{g}_j = \mathbf{g}_i$, and so on. (Eventually, each entry of the list has the form (i, \mathbf{g}_i, ctr_i) .) Then, store the list into qRAM.

Note that $0 \leq ctr_i \leq \mu - 1$ for all i , and the value ctr_i is represented as a $\log \mu$ bit string. If \mathbf{g}_i can be computed with $O(1)$ operations for each i , this precomputation can be completed with $O(N \log N)$ operations.

How to Prepare $|\psi\rangle$. We implement a unitary U satisfying $U|0^n\rangle = |\psi\rangle$ as the following algorithm⁴.

Algorithm PREP1.

1. Create the superposition $\sum_{0 \leq i \leq N-1} \sqrt{1/N} |i\rangle$.

⁴ The idea of applying Hadamard to the rightmost register in Step 4 and then using QAA is inspired from the state preparation technique by Sanders et al. [66].

2. Access qRAM to obtain $\sum_{0 \leq i \leq N-1} \sqrt{1/N} |i\rangle |\mathbf{g}_i\rangle |ctr_i\rangle$. (By abuse of notation, we denote \mathbf{g}_N by \mathbf{g}_0 .)
3. Multiply each basis state by the phase $(-1)^{\zeta_i}$ by accessing qRAM. Now the state is $\sum_{0 \leq i \leq N-1} \sqrt{1/N} (-1)^{\zeta_i} |i\rangle |\mathbf{g}_i\rangle |ctr_i\rangle$.
4. Set the leftmost register to $|0^\ell\rangle$. This is done by searching for the tuple (\mathbf{g}_i, ctr_i) in qRAM and adding the corresponding index to the first register. The resulting state is $|0^\ell\rangle \sum_{0 \leq i \leq N-1} \sqrt{1/N} (-1)^{\zeta_i} |i\rangle |\mathbf{g}_i\rangle |ctr_i\rangle$.
5. Apply the Hadamard gates to the rightmost register. Some calculations show that the resulting state is

$$\frac{1}{\sqrt{N}\sqrt{\mu}} |0^\ell\rangle \left(\sum_{\mathbf{w}} \Psi(\mathbf{w}) |\mathbf{w}\rangle \right) |0^{\log \mu}\rangle + |\varepsilon\rangle, \tag{12}$$

where the third register of $|\varepsilon\rangle$ is orthogonal to $|0^{\log \mu}\rangle$.

6. Apply QAA (that returns a correct answer with certainty) on (12) to amplify the state of which the third register is $0^{\log \mu}$. This can be done by performing Steps 1–5 and their uncomputations at most $p_{\text{init}}^{-1/2}$ times each in total, where $p_{\text{init}} := \sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2 / N\mu$.

The complexity of PREP1 depends on G and ζ but it is small in most cases.

For example, suppose $\mathbf{g}_i \neq \mathbf{g}_j$ holds for $i \neq j$. Then $\mu = 1$ and $\sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2 = N$ hold, which implies $p_{\text{init}} = 1$. In particular, QAA is actually not necessary and a single execution of Steps 1–5 is sufficient to prepare $|\psi\rangle$.

Even if ζ and each \mathbf{g}_i are random, the number of QAA iterations in Step 6 is at most $O(\ell / \log \ell)$ on average: Since \mathbf{g}_i is random, $\mu \leq \ell / \log \ell$ holds with an overwhelming probability by the standard balls-into-bins arguments [61, Lem. 5.12]. In addition, the value $|\Psi(\mathbf{w})|^2 = \left| \sum_{i: \mathbf{g}_i = \mathbf{w}} (-1)^{\zeta_i} \right|^2$ is always a non-negative integer, and $\Pr [|\Psi(\mathbf{w})|^2 \neq 0] \geq 1/2$ holds for each \mathbf{w} because ζ is random. Hence, the expected value $\sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2$ is at least $\#\{\mathbf{w} : \Psi(\mathbf{w}) \neq 0\} \times (1/2) \geq N/2\mu$, and $p_{\text{init}} = \sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2 / N\mu \geq 1/(2\mu^2) \gtrsim (\log \ell)^2 / (2\ell^2)$ on average.

The Entire Algorithm and Analysis. Our Q1 attack runs as follows.

Algorithm QFCA1.

1. Get a keystream segment long enough to mount the attack.
2. Perform the precomputation described on p.15.
3. Run QAAw/oKp on p.8 with $U := H^{\otimes \ell} \cdot \text{PREP1}$ on the Boolean function $f : \mathbb{F}_2^\ell \rightarrow \mathbb{F}_2$ such that $f(\mathbf{x}) = 1$ iff $\mathbf{x} = \boldsymbol{\sigma}^{(0)}$. (How to compute f depends on attack targets.)

Let p be the probability that we obtain an \mathbf{x} satisfying $f(\mathbf{x}) = 1$ when we measure the state $H^{\otimes \ell} \cdot \text{PREP1}|0^\ell\rangle (= H^{\otimes \ell}|\psi\rangle)$. That is,

$$p = \frac{N^2 \cdot \text{Cor}(\boldsymbol{\sigma}^{(0)}G, \zeta)^2}{2^\ell \cdot \sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2}. \tag{13}$$

By the claim at the end of Sect. 3.3, with probability at least 0.9 (when K and IV are randomly chosen and ℓ is sufficiently large, e.g., $\ell \geq 10$), $\text{Cor}(\mathbf{x}G, \boldsymbol{\zeta})^2 \leq c^2/4$ holds for all $\mathbf{x} \neq \boldsymbol{\sigma}^{(0)}$ and $\text{Cor}(\boldsymbol{\sigma}^{(0)}G, \boldsymbol{\zeta})^2 \geq c^2/2$ holds. Provided these inequalities really hold,

$$p \geq \frac{N^2 c^2}{2^{\ell+1} \cdot \sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2} \tag{14}$$

holds, and the attack finds the correct decoding result $\boldsymbol{\sigma}^{(0)}$ in expected time complexity at most about

$$T_{\text{total}} = T_{\text{precomp}} + (9/2)p^{-1/2} (2 \cdot T_{\text{prepare}} + T_f), \tag{15}$$

where T_{prepare} (resp., T_f) is the running time of the algorithm PREP1 (resp., the time complexity required to compute f). In addition, T_{precomp} is the time complexity to collect necessary data and perform the precomputation. How to compute f depends on the internal structure of the target cipher.

Typically, we have $T_{\text{prepare}} \ll T_f$, $T_{\text{precomp}} = O(N)$, and $\sum_{\mathbf{w}} |\Psi(\mathbf{w})|^2 = O(N)$, when the complexity (15) becomes roughly about $(N + T_f \cdot \frac{2^{\ell/2}}{\sqrt{Nc^2}})$. Balancing the two terms, we obtain

$$N = 2^{\ell/3} \cdot (T_f/c)^{2/3}. \tag{16}$$

In summary, with probability at least 0.9 (on the randomness of K and IV), the attack recovers $\boldsymbol{\sigma}^{(0)}$ in expected time complexity $2^{\ell/3+1} \cdot (T_f/c)^{2/3}$.

4.3 Discussions and Observations

The above algorithm QFCA1 is a very natural extension of classical fast correlation attacks. By applying QFCA1 to speed up existing classical fast correlation attacks, we expected to achieve quantum attacks faster than the Grover search. However, we have not obtained a meaningful speedup so far with this approach.

One reason is that the absolute correlations in some classical attacks are too small. For instance, the attack on Grain v1 by Todo et al. [75] utilizes linear approximations of absolute correlation $c = 2^{-36}$, while both the LFSR and key lengths of Grain v1 are 80. If a single linear approximation is used, we need the data complexity at least $c^{-2} \geq 2^{72}$, which is much larger than the exhaustive key search with Grover’s algorithm. The data complexity (and time complexity possibly also) decreases to some extent by using multiple approximations, but we find it still hard to achieve an attack faster than the Grover search.

Another reason is that the LFSR length is quite large in some ciphers. For instance, SNOW 2.0 [28] is based on a 512-bit LFSR. As explained around Eq. (16), we will have a factor of order $2^{\ell/2}$ or $2^{\ell/3}$ in the time complexity, which is too large when $\ell = 512$. Classical attacks reduce the dimension of the code (i.e., the parameter ℓ) by solving k -sum problems. However, in the Q1 setting, we observe that the cost to solve k -sum problems sufficiently reducing the dimension is too heavy (even with the dedicated quantum algorithms [38, 63, 68]) compared to the quantum exhaustive key search with Grover’s algorithm when

k is small (e.g., $k = 2$), and the correlations after dimension reduction become too small when k is large (e.g., $k = 4$).

Due to these reasons, we suspect it is quite hard to mount a fast correlation attack that is faster than the generic attack in the Q1 setting, or more fairly non-trivial techniques will be required. Given this situation, we next focus on Q2 attacks.

Remark 1. The state $H^{\otimes \ell}|\psi\rangle$ in Eq. (11) is a superposition of candidate messages $|\mathbf{x}\rangle$ and the quantum amplitude is proportional to the correlation $\text{Cor}(\mathbf{x}G, \zeta)$ and so can be regarded as an analogy of the *correlation state* in Schrottenloher’s quantum linear key recovery. Still, our technique and Schrottenloher’s are quite different. Unlike the preparation of the correlation state, convolutions are not computed in preparing $H^{\otimes \ell}|\psi\rangle$. Moreover, in the Q2 attack in Sect. 6, we will use Shor’s algorithm to efficiently prepare a state corresponding to $H^{\otimes \ell}|\psi\rangle$.

Remark 2. Measuring the state (11), we obtain \mathbf{x} with a probability proportional to $|\mathcal{W}(\Psi)(\mathbf{x})|^2$. This can be regarded as a random sampling according to the distribution induced by $\mathcal{W}(\Psi)$. A possible alternative approach could be to iteratively perform this sampling and estimate the values $|\mathcal{W}(\Psi)(\mathbf{x})|^2$ instead of applying QAA, but so far, we have not found more efficient attacks with this idea. Leveraging such random samplings in a better way could be a possible future research to investigate.

Remark 3. The attacks in this section essentially rely on linear approximations of which the linear masks cover the entire state of LFSR. If one were to use masks that only cover part of LFSR (as done in, e.g., [7]), the attacks would proceed by applying our method for the part covered by the linear masks and guessing the remaining part with the Grover search. The same thing holds true for the Q2 attacks shown later.

5 New Attack Model and Security Definition in Q2

This section introduces a new attack model and a security definition for stream ciphers in the Q2 setting.

When studying Q2 attacks, we must carefully consider which attack breaks what security notion. This is because the Q2 setting allows adversaries to perform operations that were never anticipated when some classical security notions were defined, e.g., querying all the messages at once in superposition. Let us briefly illustrate this with attacks on MACs as an example. A classical attack on a deterministic MAC is considered meaningful if it forges a valid tag for a message that has not been queried by the adversary. Meanwhile, Q2 attacks are typically allowed to query all messages simultaneously in quantum superposition. This makes it unclear how we should interpret the meaning of a Q2 attack on a MAC if it produces several valid message-tag pairs after making a single quantum query consisting of exponentially many messages in superposition. The seminal work

by Kaplan et al. [48] carefully addresses this issue and demonstrates that (some of) their attacks on MACs are valid in that they break Boneh and Zhandry’s EUF-qCMA security [10].

As we will see below, there is also a subtle issue regarding Q2 attacks on stream ciphers. In what follows, We denote a random function by RF, of which the domain and range will be clear from the context.

Classical Security Notion: IV-Based Stream Ciphers as PRFs. As shown by Berbain and Gilbert [6], the classical security definition appropriate for IV-based stream ciphers is the Pseudo-Random Function (PRF) security. Here, stream ciphers are regarded as keyed functions $SC : \mathbb{F}_2^\kappa \times \mathbb{F}_2^{iv} \rightarrow \mathbb{F}_2^D$ that take key and IV as input and return a keystream of length D for some $D \gg 1$. Recall that the PRF advantage of an oracle-aided algorithm \mathcal{A} for SC is defined as

$$\text{Adv}_{SC}^{\text{PRF}}(\mathcal{A}) := |\Pr[\mathcal{A}^{\text{SC}^\kappa} \text{ outputs } 1] - \Pr[\mathcal{A}^{\text{RF}} \text{ outputs } 1]|, \quad (17)$$

where the probability is taken over both the randomness of \mathcal{A} and the choice of the secret key K or the random function RF. The ciphers are considered secure iff no adversary \mathcal{A} with reasonable computational resources can distinguish SC and RF with a non-negligible advantage.

qPRF Security and Some Issues. The counterpart of the PRF security in the Q2 setting is the quantum pseudo-random function (PRF) security by Zhandry [82], where the oracle of the keyed function and the random function are replaced with the corresponding quantum oracles that accept inputs and returns outputs in quantum superposition. Thus, to choose a security definition for stream ciphers in the Q2 setting, the easiest way is simply to adapt the qPRF security.

However, the qPRF security does not mesh well with stream ciphers. Typical Q2 attacks assume a moderate (polynomial) size quantum computer with qRAM, whereas the quantum oracle of stream ciphers returns an exponentially long output in quantum superposition all at once. In other words, a quantum computer of a moderate (e.g., 2^{20}) size has a register of a very large (e.g., 2^{60}) size to receive outputs from the oracle, which is quite unbalanced. A potential solution to this problem is to limit the output length of oracles, but this overlooks one of the primary features of stream ciphers, which is that a long keystream can be generated from a single IV. An alternative solution could be to assume that the oracle’s outputs are written into qRAM, but this approach would require a substantial amount of operations for adversaries just to read the oracle’s outputs. It undermines the meaning of studying Q2 attacks because unexpectedly efficient and intriguing Q2 attacks are usually achieved by efficiently processing a superposition of many outputs from an oracle.

qBPRF Security. To remedy this, we introduce a new security definition, which we call the *quantum Booleanized PRF security*, or qBPRF security for short.

First, let us define the *Booleanization* of a function $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ as the Boolean function $BF : \mathbb{F}_2^n \times \mathbb{F}^{\log m} \rightarrow \mathbb{F}_2$ such that $BF(x, i) = (F(x))_i$, and the quantum Booleanized PRF (qBPRF) security as the qPRF security of BF .

We define the qBPRF advantage of an algorithm \mathcal{A} for a stream cipher $SC : F_2^\kappa \times F_2^{iv} \rightarrow F_2^D$ as the qPRF advantage of its Booleanization, namely,

$$\text{Adv}_{SC}^{\text{qBPRF}}(\mathcal{A}) := \left(\text{Adv}_{BSC}^{\text{qPRF}}(\mathcal{A}) = \right) \left| \Pr \left[\mathcal{A}^{BSC^\kappa} \text{ outputs } 1 \right] - \Pr \left[\mathcal{A}^{\text{RF}} \text{ outputs } 1 \right] \right|,$$

where \mathcal{A} is allowed to make quantum queries to the oracles. We say that the stream cipher SC is qBPRF-secure if no adversary faster than the generic attack can have a non-negligible qBPRF advantage.

Put differently, we regard an attack on the cipher breaks its qBPRF security if its computational cost is less than the generic attack with Grover’s algorithm while the qBPRF advantage is close to 1, and we aim to find such (fast correlation) attacks in the next section. In particular, we assume that the quantum oracle of the Booleanized version of the target cipher is given to an adversary. By considering the Booleanized versions, we can keep the output length of the oracle short while taking long keystreams into account, addressing the aforementioned issues. To prevent trivial attacks, we set an appropriate limit on D , of which the details will be discussed later.

Feasibility. The attack model in the definition of qBPRF security is quite strong because it essentially assumes an adversary can query not only IVs but also indices for keystream bits in quantum superposition. Yet, we argue that qBPRF security is worth studying and feasible in that some stream ciphers based on the CTR mode, e.g., (some members of) Salsa20 [9] and ChaCha [8] families, seem to achieve it. Below, we explain this by showing a security reduction.

Let $F : \mathbb{F}_2^\kappa \times \mathbb{F}_2^{iv} \times \mathbb{F}_2^{\text{ctr}} \rightarrow \mathbb{F}_2^m$ be a keyed function where \mathbb{F}_2^κ is the key space, and $D' \gg 1$ be a parameter. Let $\text{CTR}^F : \mathbb{F}_2^\kappa \times \mathbb{F}_2^{iv} \rightarrow \mathbb{F}_2^{D' \cdot m}$ be the stream cipher generating a keystream segment as

$$\text{CTR}_K^F(IV) := F_K(IV, 0) || F_K(IV, 1) || \cdots || F_K(IV, D')$$

Then, the following proposition holds.

Proposition 4. *Suppose $D' < 2^{iv}$. For any quantum algorithm \mathcal{A} making q queries, there is another quantum algorithm \mathcal{B} making q quantum queries such that*

$$\text{Adv}_{\text{CTR}^F}^{\text{qBPRF}}(\mathcal{A}) \leq \text{Adv}_F^{\text{qPRF}}(\mathcal{B}),$$

where the time, memory complexity, and qubits required to run \mathcal{B} are at most $O(1)$ times larger than those for \mathcal{A} .

Proof. We construct \mathcal{B} so that it simply emulates the oracle for \mathcal{A} by using the one given to itself. That is, when \mathcal{A} queries a pair (IV, i) , \mathcal{B} queries $(IV, \lfloor i/m \rfloor)$ to its own oracle, truncating the response y from the oracle and sending the $(i - m \cdot \lfloor i/m \rfloor)$ -th bit of y to \mathcal{A} . (Note that arbitrary bit of y can be computed in quantum superposition by making only a single query to \mathcal{B} ’s oracle [43].) Using this \mathcal{B} , the claim of the proposition obviously holds. \square

Roughly speaking, the above proposition states that CTR^F is qBPRF-secure as long as there is no attack breaking the qPRF security of F (as long as $D' < 2^{iv}$). ChaCha and Salsa20 families adapt the structure of the above CTR for some F , and there have been reported no Q2 attacks distinguishing their underlying function F faster than the Grover search. Therefore, some of these ciphers, including Salsa20/12, Salsa20/20, and ChaCha20, will likely achieve the qBPRF security⁵.

Upper Limit of Keystream Bit Index. When studying attacks to break the qBPRF security of a stream cipher $\text{SC} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^{iv} \rightarrow \mathbb{F}_2^D$, we must set an appropriate upper limit for the keystream bit index, i.e., the parameter D , to prevent trivial attacks. For example, if the $(i + j)$ -th bit of each keystream is always equal to the i -th bit for some exponentially large j , the parameter D should be less than j . Otherwise, an adversary can efficiently break the qBPRF security by, e.g., getting the first κ bits and $(j + 1)$ -th, \dots , $(j + \kappa)$ -th bits of a keystream and check whether they are equal.

When studying LFSR-based stream ciphers, we set D to be the period of the underlying LFSR, which is $2^\ell - 1$ if LFSR's bit length is ℓ . This may exceed data limits specified by the designers of a target cipher. Still, considering that even in the classical setting, the first step is to show an attack exceeding the designers' limit (e.g., [71]), we set the limit D as large as possible in the quantum setting.

Remarks. The oracle of the Booleanized version of a stream cipher enables an adversary to efficiently get the i -th bit of a keystream for arbitrarily large i (as long as i is smaller than an appropriately set upper limit). Some readers may be concerned that such oracles may significantly speed up some attacks even in the *classical* setting. Indeed, if such a classical oracle is available, the *data* complexity of some classical fast correlation attacks will be reduced to some extent because only specific bits of keystream segments are used [76]. However, we expect that the *time* complexity of fast correlation attacks will not be significantly affected because the decoding algorithms do not care much whether the size of the indices i involved in decoding procedures is large or not.

Our primary objective of studying attacks on qBPRF security is to uncover interesting properties and deepen our understanding of the power of Q2 attacks. We do not claim that the practical security of a scheme is affected, even if we discover an efficient attack that only compromises the qBPRF security. We argue that it is worthwhile to study attacks on qBPRF security because we can obtain an interesting new type of large quantum speed-up for fast correlation attacks on some LFSR-based stream ciphers, while some other stream ciphers including Salsa20/12, Salsa20/20, and ChaCha20 are almost completely intact, as we showed around Proposition 4.

⁵ Some members of the families, e.g., Salsa20/8, have already been broken in the classical setting [3], but we are unsure whether they can be converted into a Q2 attack faster than the Grover search.

Often, quantum attacks breaking a rather theoretical security notion do not immediately imply attacks that compromise more practical security notions. Still, some of such attacks later have become the indispensable basis of other attacks with much more practical implications. For example, the Q2 attacks on the Even-Mansour and FX constructions [51, 52] paved the way for the technique to exponentially reduce memory complexity in some Q1 attacks by using Simon’s algorithm [12] and achieving a more-than-quadratic speed-up in the Q1 model [14]. The subsequent sections present attacks on the qBPRF security, hoping they will serve as the foundation for even more impactful attacks.

6 Quantum Fast Correlation Attack in the Q2 Model

6.1 Overview and Rough Idea

When mounting fast correlation attacks in the Q2 model, we aim to break the qBPRF security of a target stream cipher, assuming that the Booleanized version of the cipher is given as a quantum oracle.

The oracle allows an adversary to query IVs in quantum superposition as well as indices of keystreams, but we fix an arbitrarily chosen IV through an attack like in the classical and Q1 settings. Namely, the primary goal of the attack is to recover the initial state of an LFSR for a single pair of the key and an IV, and we make superposition queries only for keystream indices. The basic idea of the attack is the same as in Sect. 4.1. That is, we (i) prepare the quantum state $|\psi\rangle$ of Eq. (10), (ii) apply the Hadamard transform on $|\psi\rangle$, and then (iii) apply QAA to amplify the quantum amplitude of the correct decoding result $|\sigma^{(0)}\rangle$.

The difference is as follows.

- We focus on decoding problems derived from a single linear approximation as explained below Eq. (7). In particular, the parameter ℓ is equal to the bit length of the LFSR, G is a matrix of which the i -th column is $M^{i-1} \cdot \mathbf{I}$ for some \mathbf{I} (recall that M is the LFSR’s state update matrix) and a decoding algorithm returns $\sigma^{(0)} = \mathbf{s}^{(0)}$, the initial state of the LFSR.
- We set the parameter N (the number of columns of G) to be $2^\ell - 1$, the period of the LFSR. This implies that G is an $\ell \times (2^\ell - 1)$ matrix over \mathbb{F}_2 . At first glance, it might seem that this would make the preparation of $|\psi\rangle$ prohibitively costly. However, our core observation is that $|\psi\rangle$ can be prepared quite efficiently by regarding the state update of LFSR as multiplication in a finite field and applying Shor’s algorithm for the discrete logarithm problem to find the index i satisfying $\mathbf{g}_i = \mathbf{x}$ for a given \mathbf{x} . (The main reason for only focusing on G derived from a single linear approximation is to utilize this technique.)

- The Boolean function $f(\mathbf{x})$ is computed by checking whether the value $\text{Cor}(\mathbf{x}G, \boldsymbol{\zeta})^2$ is above a certain threshold by using the quantum counting algorithm, like Kaplan et al. [49] did for quantum linear distinguishers⁶. We do not use methods depending on the structure of target ciphers because the method with the quantum counting algorithm is the most efficient for all the applications we have found so far.

As the structure of G is restricted and we compute f independently from the structure of target ciphers, our Q2 attack can be formulated as an algorithm to solve the following general problem.

Problem 1. Let M be the state update matrix (Eq. (2)) of an LFSR of length L over \mathbb{F}_{2^n} and G be an $\ell \times (2^\ell - 1)$ matrix over \mathbb{F}_2 of which the i -th column vector is $M^{i-1} \cdot \boldsymbol{\Gamma}^\top$ for some $\boldsymbol{\Gamma} \in \mathbb{F}_2^\ell (= \mathbb{F}_{2^n}^L)$, where $\ell := n \cdot L$. Let $\mathbf{s}^{(0)}$ be a vector in \mathbb{F}_2^ℓ and $\boldsymbol{\zeta}$ be a binary sequence of length $2^\ell - 1$ defined as $\boldsymbol{\zeta} := (\mathbf{s}^{(0)}G) \oplus \mathbf{e}$, where the bits of $\mathbf{e} = (e_0, \dots, e_{2^\ell-2})$ are independently and randomly chosen in such a way that $\Pr[e_i = 1] = p$ for all i , with $p \approx (1 + c)/2$ or $(1 - c)/2$ for some $c > 0$. Given the quantum oracle of the Boolean function that returns ζ_i on an input $i \in \{0, \dots, 2^\ell - 2\}$, compute $\mathbf{s}^{(0)}$.

The next subsection presents a quantum algorithm to solve this problem.

6.2 Formal Details

We first explain how to compute f , and then show an algorithm to prepare $|\psi\rangle$. After that, we describe the entire algorithm and provide analysis.

We denote the i -th column vector of G by \mathbf{g}_i as before and put $N := 2^\ell - 1 (= 2^{nL} - 1)$. Note that $\mathbf{g}_i \neq \mathbf{g}_j$ for $i \neq j$ and that $\{\mathbf{g}_i\}_{1 \leq i \leq N} = \mathbb{F}_2^\ell \setminus \{\mathbf{0}\}$ ($= \mathbb{F}_2^{nL} \setminus \{\mathbf{0}\}$) follows from the definition of G and Eq. (5). In particular, we have that $\Psi(\mathbf{g}_i) = (-1)^{\zeta_i}$ for all i and $\sum_{\mathbf{w} \neq \mathbf{0}} |\Psi(\mathbf{w})|^2 = 2^\ell - 1 (= N)$, which implies

$$|\psi\rangle = \sum_{1 \leq i \leq N} \frac{(-1)^{\zeta_{i-1}}}{\sqrt{N}} |\mathbf{g}_i\rangle.$$

In what follows, we use these properties without any mention.

Computing a Boolean Function for QAA by Quantum Counting. Here we explain how to compute $f(\mathbf{x})$ such that $f(\mathbf{x}) = 1$ iff $\mathbf{x} = \mathbf{s}^{(0)}$.

Define a Boolean function f' by $f'(\mathbf{x}) = 1$ iff $\text{Cor}(\mathbf{x}G, \boldsymbol{\zeta})^2 \geq 3c^2/8$. Then, the claim at the end of Sect. 3.3 ensures $\Pr_{K,IV} [f(\mathbf{x}) = f'(\mathbf{x}) \text{ for all } \mathbf{x}] \geq 0.9$.

⁶ Saying it differently, we prepare a superposition of $|\mathbf{x}\rangle$ with the amplitude being proportional to the correlation $\text{Cor}(\mathbf{x}G, \boldsymbol{\zeta})$, and then amplify the “good” \mathbf{x} by computing the correlation again and checking whether it is large enough. The idea of using a single value for both preparation and amplification is not new and has already appeared in, e.g., [4].

With this in mind, we implement the unitary operator $\mathcal{S}_{f'}$ satisfying $\mathcal{S}_{f'}|\mathbf{x}\rangle = (-1)^{f'(\mathbf{x})}|\mathbf{x}\rangle$. To implement this, we count the number of i satisfying $(\mathbf{x}G)_i = \zeta_i$ for each $i = 0, 1, \dots, N - 1$ by using the quantum counting algorithm with a sufficiently high precision. Proposition 3 ensures that the error probability of the quantum counting algorithm is as small as 0.2, but this is still large if the algorithm is used in QAA as a subroutine. To make the error probability small enough, we run multiple instances and perform a majority vote.

Concretely, to implement $\mathcal{S}_{f'}$, we run the following algorithm. Here, r is a parameter fixed later, and $h_{\mathbf{x}} : \{0, \dots, N\} \rightarrow \mathbb{F}_2$ is the Boolean function defined⁷ by $h_{\mathbf{x}}(i) = (\mathbf{x}G)_i \oplus \zeta_i \oplus 1$ for $0 \leq i \leq N - 1$ and $h_{\mathbf{x}}(N) = 0$.

Algorithm JDG.

0. (Assume a basis state $|\mathbf{x}\rangle$ is given as an input.)
1. For $j = 1, \dots, r$, perform the following procedure.
 - (a) Run the quantum counting algorithm (QC on page 8) with $q = 2^7/c$ to compute an estimation of $Z := |h_{\mathbf{x}}^{-1}(1)|$. Let \tilde{Z}_j be the resulting output.
 - (b) Let $\tilde{C}_j := \frac{2\tilde{Z}_j - N}{N}$. Compute $(\tilde{C}_j)^2$ and write it into a new auxiliary register.
 - (c) Uncompute Step (a).
2. Check whether at least $r/2$ values among $(\tilde{C}_1)^2, \dots, (\tilde{C}_r)^2$ are greater than or equal to $3c^2/8$. If so, multiply the entire state by the phase (-1) . Otherwise, do nothing.
3. Uncompute Step 1.

Proposition 5. *Assume $\ell \geq 10$, $\text{Cor}(\mathbf{x}G, \zeta)^2 \leq c^2/4$ holds for all $\mathbf{x} \neq \mathbf{s}^{(0)}$, and $\text{Cor}(\mathbf{s}^{(0)}G, \zeta)^2 \geq c^2/2$. By abuse of notation, let JDG also denote the unitary operator corresponding to the above algorithm. Then, $f = f'$ holds, and the operator norm of $(\text{JDG} - \mathcal{S}_f)$ is upper bounded as $\|\text{JDG} - \mathcal{S}_f\|_{\text{op}} \leq 2^{(\ell/2) - (0.1r) + 1}$. The depth required to implement JDG on a quantum circuit is at most about $2^{11}r\ell^3/c$, and JDG makes $2^{8r}/c$ queries to the oracle.*

See Section D of the full version of this paper [42] for a proof. We will set $r = 25\ell$ such that the difference $\|\text{JDG} - \mathcal{S}_f\|_{\text{op}}$ is extremely small ($\leq O(2^{-2\ell})$) and we can use JDG instead of \mathcal{S}_f in QAA while keeping the success probability almost unchanged.

The amount of auxiliary qubits required for JDG is at most the maximum of

- the qubits needed to compute $h_{\mathbf{x}}$,
- the qubits needed to perform the classical computation in Step 2,
- the qubits needed for Calc on page 8,

which is in $O(\ell^2)$. (See Section E of the full version of this paper [42] for details on the qubits required for $h_{\mathbf{x}}$. For Calc, we assume that the sin function is approximately computed using a constant number of terms in the Taylor expansion.)

⁷ $h_{\mathbf{x}}$ is defined so that $h_{\mathbf{x}}(i) = 1$ iff $(\mathbf{x}G)_i = \zeta_i$ for $i < N$. The domain size is set as $(N + 1)$ instead of N to make it a power of two.

How to Prepare $|\psi\rangle$. Roughly speaking, we prepare $|\psi\rangle$ by (i) making a superposition of all $\mathbf{x} \in \mathbb{F}_2^\ell \setminus \{\mathbf{0}\}$, (ii) compute i such that the i -th column of G (denoted by \mathbf{g}_i) is equal to \mathbf{x} , and (iii) multiply the phase $(-1)^{\zeta_{i-1}}$ by querying to the quantum oracle.

To perform (ii), we utilize Shor’s algorithm and the relationship between the state update of LFSR and multiplication in the finite field. Let ξ be the isomorphism defined in Eq. (3). The most important observation is that we have

$$i = \log_\alpha (\xi(\mathbf{g}_i)/\xi(\mathbf{\Gamma})) + 1 = \log_\alpha (\xi(\mathbf{g}_i)) - \log_\alpha (\xi(\mathbf{\Gamma})) + 1 \tag{18}$$

for each i because

$$\xi(\mathbf{g}_i) = \xi(\mathbf{\Gamma}(M^\top)^{i-1}) \stackrel{\text{Eq. (4)}}{=} \xi(\mathbf{\Gamma}) \cdot \alpha^{i-1},$$

holds. Therefore, we can compute i such that $\mathbf{g}_i = \mathbf{x}$ for a given $\mathbf{x} \in \mathbb{F}_2^\ell \setminus \{\mathbf{0}\}$ as $i = \log_\alpha (\xi(\mathbf{g}_i)) - \log_\alpha (\xi(\mathbf{\Gamma})) + 1$, applying Shor’s discrete logarithm in the multiplicative group $\mathfrak{F}^\times = (\mathbb{F}_{2^n}[x]/(f^*(x)))^\times \cong \mathbb{Z}/N\mathbb{Z}$.

First, we describe our algorithm when a unitary operator DLOG satisfying

$$\text{DLOG}|\mathbf{x}\rangle|0\rangle = |\mathbf{x}\rangle|\log_\alpha(\mathbf{x})\rangle$$

is available as a quantum circuit. After that, we discuss the complexity to approximate it with sufficiently high precision by using Shor’s algorithm.

Algorithm PREP2.

1. Prepare the superposition

$$\sum_{\mathbf{x} \in \mathbb{F}_2^\ell \setminus \{\mathbf{0}\}} \frac{1}{\sqrt{N}} |\mathbf{x}\rangle.$$

2. For each basis state \mathbf{x} , run DLOG twice to compute $\log_\alpha(\xi(\mathbf{x}))$ and $\log_\alpha(\xi(\mathbf{\Gamma}))$. Then compute $i := \log_\alpha(\xi(\mathbf{x})) - \log_\alpha(\xi(\mathbf{\Gamma})) + 1$. Now the state is

$$\sum_{\mathbf{x} \in \mathbb{F}_2^\ell \setminus \{\mathbf{0}\}} \frac{1}{\sqrt{N}} |\mathbf{x}\rangle |\log_\alpha(\xi(\mathbf{x}))\rangle |\log_\alpha(\xi(\mathbf{\Gamma}))\rangle |i\rangle.$$

3. By querying $(i - 1)$ to the oracle, multiply the entire state by the phase $(-1)^{\zeta_{i-1}}$ (Recall that now we are assuming the oracle that returns ζ_i for the input i in quantum superposition.)
4. Uncompute Step 2. Now, the state is $|\psi\rangle$.

Both the T -depth and the number of ancillary qubits are dominated by Step 2 (and its uncomputation). By running 8ℓ instances of Shor’s algorithm DLOG can be approximated with error (w.r.t. operator norm) in $O(2^{-2\ell})$, T -depth at most $2^8\ell^3 + O(\ell^2)$, and $O(\ell^2)$ ancillary qubits (see Section F of the full version of this paper [42] for details). Therefore, PREP2 can be implemented with error in $O(2^{-2\ell})$, T -depth $2^9\ell^3 + O(\ell^2)$, and $O(\ell^2)$ ancillary qubits.

The Entire Algorithm and Analysis. Our algorithm solving Problem 1 runs as follows.

Algorithm QFCA2. Run QAAw/oKp on p.8 with $U := H^{\otimes \ell} \cdot \text{PREP2}$ on the Boolean function $f : \mathbb{F}_2^\ell \rightarrow \mathbb{F}_2$ such that $f(\mathbf{x}) = 1$ iff $\mathbf{x} = \mathbf{s}^{(0)}$. Here, f is computed by using the algorithm JDG. The parameter r for JDG is chosen as $r := 25\ell$.

Below we analyze the complexity of QFCA2 assuming $\ell \geq 10$ (so that the assumption of Proposition 5 will be satisfied) and $c \leq \ell^{-1}$, which is the case for the applications we will see later.

Let p be the probability that we obtain an \mathbf{x} satisfying $f(\mathbf{x}) = 1$ when we measure the state $H^{\otimes \ell} \cdot \text{PREP2}|0^\ell\rangle (= H^{\otimes \ell}|\psi\rangle)$. That is,

$$p = \frac{N^2 \cdot \text{Cor}(\mathbf{s}^{(0)}G, \zeta)^2}{2^\ell \cdot \sum_w |\Psi(\mathbf{w})|^2} = \frac{2^\ell}{2^\ell - 1} \cdot \text{Cor}(\mathbf{s}^{(0)}G, \zeta)^2. \tag{19}$$

By the claim at the end of Sect. 3.3, $\text{Cor}(\mathbf{x}G, \zeta)^2 \leq c^2/4$ holds for all $\mathbf{x} \neq \mathbf{s}^{(0)}$, with probability at least 0.9 (when K and IV are randomly chosen). Provided this condition hold,

$$p \geq \frac{2^\ell}{2^\ell - 1} \cdot c^2 \approx c^2 \tag{20}$$

follows from Eq. (19), and the attack finds the correct decoding result $\mathbf{s}^{(0)}$ in expected time complexity at most about

$$T_{\text{total}} = (9/2)p^{-1/2} (2 \cdot T_{\text{prepare}} + T_f) \lesssim (9/2) \frac{1}{c} (2 \cdot T_{\text{prepare}} + T_f) \tag{21}$$

where T_{prepare} (resp., T_f) is the running time of the algorithm PREP2 (resp., JDG). Since we set $r = 25\ell$ for JDG,

$$T_f \lesssim 25 \cdot 2^{11} \ell^4 / c$$

follows from Proposition 5. In addition, $c \leq \ell^{-1}$ is assumed. Meanwhile, as discussed before, $T_{\text{prepare}} = 2^9 \ell^3 + O(\ell^2)$ holds. Hence $T_{\text{prepare}} \ll T_f$ holds and the total time complexity can be estimated as

$$T_{\text{total}} \lesssim (9/2) \cdot \frac{1}{c} \cdot (25 \cdot 2^{11} \ell^4 / c) \leq 2^{18} \cdot \ell^4 / c^2. \tag{22}$$

In addition, since PREP2 makes only $O(1)$ queries, the number of queries made by QFCA2 can be approximated by the number of queries made by JDG multiplied by the number of applications of JDG. Therefore, the number of quantum queries made by QFCA2 is at most about $(9/2)p^{-1/2} \cdot (2^8(25\ell)/c) \lesssim 2^{15} \ell / c^2$.

Next, we analyze the success probability. Failure of QFCA2 is attributed to the following four factors:

- (1) Whether the assumption of Proposition 5 about the correlations (i.e., the assumption of the claim at the end of Sect. 3.3) is satisfied or not.

- (2) The error in JDG approximating \mathcal{S}_f (provided the assumption of Proposition 5 is satisfied).
- (3) The error in approximating DLOG.
- (4) Failure of QAA to find the correct value $\mathbf{s}^{(0)}$.

The error probability coming from (1) is at most 0.1 (because of the claim at the end of Sect. 3.3 and the assumption $\ell \geq 10$), and (4) is already taken into account in the expected complexity of QAA shown in Proposition 2. Hence, if (2) and (3) could be ignored, the algorithm would successfully recover $\mathbf{s}^{(0)}$ in the expected time complexity shown in Eq. (22) with a probability of at least 0.9 (with respect to the randomness of K and IV).

Regarding (2), the distance between JDG and \mathcal{S}_f is at most $O(2^{-2\ell})$. This means that the failure probability of QAA with t applications of \mathcal{S}_f increases by $O(t \cdot 2^{-2\ell})$ if \mathcal{S}_f is replaced with JDG. The same holds for the approximation of DLOG. As the overall complexity of QFCA2 is $O(\ell^4 c^{-2})$, the success probability decreases by $O(\ell^4 c^{-2} 2^{-2\ell})$ in total when (2) and (3) are taken into account. Therefore, the success probability of the algorithm can be estimated as at least $0.9 - O(\ell^4 c^{-2} 2^{-2\ell})$.

Summary. Assuming $\ell \geq 10$ and $c \leq \ell^{-1}$, QFCA2 solves Problem 1 with time and query complexity (approximately) at most $2^{18} \ell^4 / c^2$ and $2^{15} \ell / c^2$. The probability of success is estimated as at least $0.9 - O(\ell^4 c^{-2} 2^{-2\ell})$. The number of ancillary qubits required is $O(\ell^2)$ since both JDG and (the approximation of) DLOG are implemented with $O(\ell^2)$ qubits.

In the applications below, we will only discuss the cases where the term $O(\ell^4 c^{-2} 2^{-2\ell})$ is negligibly small.

Remark 4. If QFCA2 is applied to a stream cipher with an ℓ -bit LFSR, it requires $O(\ell^2)$ qubits. Meanwhile, the exhaustive key search with Grover's algorithm would require only $O(\ell)$ qubits. Strictly speaking, the validity of a dedicated quantum attack such as QFCA2 should be compared to the parallelized Grover search using the same amount of qubits. However, $O(\ell^2)$ qubits would allow to run only $O(\ell)$ parallel instances, which yields a speed-up by a factor of at most $O(\sqrt{\ell})$. This factor is not so large as to affect the validity of the attacks considered in this paper, and the cost of the Grover search also varies depending on implementations of a target cipher. Hence, in what follows, we do not take parallelization into account.

6.3 Applications

We show applications of QFCA2 on SNOW 2.0 [28], SNOW 3G [31], and Sosemanuk [5]. Our goal is to break the qBPRF security of the ciphers when the quantum oracle of the Booleanized version of the cipher is given.

SNOW 2.0. SNOW 2.0 is a stream cipher designed by Ekdahl and Johansson [28], which is standardized by ISO/IEC [45]. It consists of an LFSR of

length $L = 16$ over $\mathbb{F}_{2^{32}}$ (512 bits long in total) and a finite state machine that keeps 64-bit states. The state update and keystream generation are carried out in 32-bit words. The cipher outputs a 32-bit keystream segment at each clock, updating the internal state registers. (see Fig. 2). The key length is either 128 or 256 bit, and IVs are 128 bits. In the initialization phase, a key and an IV are linearly expanded and loaded to the registers and then mixed by updating the states 32 times, with the output bits fed back to the LFSR.

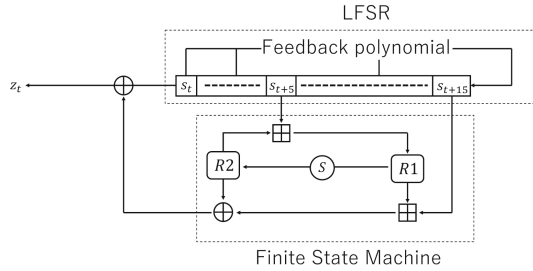


Fig. 2. SNOW 2.0. Each line corresponds to a 32-bit word. $R1$ and $R2$ are additional 32-bit registers. Modular additions are denoted by \boxplus . The circled “S” at the center is a non-linear permutation.

Linear Approximations and Classical Attacks. In the classical setting, many (linear attacks and) fast correlation attacks have been proposed on SNOW 2.0 [32,36,37,53,65,79,84]. Among others, [32,35,36] found linear approximation

$$\langle \mathbf{s}^{(t)}, \mathbf{\Gamma} \rangle_{\mathbb{F}_2} \approx \langle z_t, \mathbf{A}_1 \rangle_{\mathbb{F}_2} \oplus \langle z_{t+1}, \mathbf{A}_2 \rangle_{\mathbb{F}_2} \tag{23}$$

for some $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{F}_{2^{32}}$ and $\mathbf{\Gamma} \in \mathbb{F}_{2^{512}}$ which holds with absolute linear correlation $2^{-14.411}$. Here, $\mathbf{s}^{(t)}$ is the state of the LFSR at clock t , and $z_t \in \mathbb{F}_{2^{32}}$ is the 32-bit word (keystream segment) output by the cipher at clock t . A recent work by Gong et al. [35] also found multiple approximations with the same absolute correlation. As far as we know, $2^{-14.411}$ is the highest (absolute) linear correlation of SNOW 2.0 that has been found so far.

The current fastest classical attack on SNOW 2.0 is the fast correlation attack in [37] that uses a few linear correlations, including the above, which recovers not only the initial state of the LFSR but also the key with about 2^{159} data and 2^{162} time complexity.

Application of QFCA2. We apply QFCA2 on the decoding problem (Problem 1) derived from the linear approximation of Eq. (23). The parameters are set as $c := 2^{-14.411}$ and $\ell := 512$, and the sequence $\zeta = (\zeta_0, \zeta_2, \dots, \zeta_{N-1})$ is defined as $\zeta_t := \langle z_t, \mathbf{A}_1 \rangle_{\mathbb{F}_2} \oplus \langle z_{t+1}, \mathbf{A}_2 \rangle_{\mathbb{F}_2}$.

Recall that we aim to break the qBPRF security. Here, we briefly review the attack model. We assume the quantum oracle of the Booleanized oracle of SNOW 2.0, which we denote by $O_{BSNOW2.0}$. Given a superposition of indices i as an input, the oracle returns the i -th bit of the keystream in quantum superposition. Since the period of LFSR is $2^\ell - 1$, the upper limit of i is set as $i < 2^\ell - 1$ to prevent trivial attacks. (The oracle also allows an adversary to query IV . However, when mounting fast correlation attacks, we choose an IV arbitrarily and fix it during the attack, as in classical attacks.)

Problem 1 (and QFCA2) assumes the quantum oracle that returns ζ_i for each i (in superposition), whereas the oracle $O_{BSNOW2.0}$ returns a keystream bit of SNOW 2.0. Thus, to apply QFCA2, we simulate the oracle of ζ_i by using $O_{BSNOW2.0}$ as follows⁸.

0. (Assume a basis state $|t\rangle$ is given as an input)
1. Query $t, t+1, \dots, t+63$ to $O_{BSNOW2.0}$ to obtain $z_t, z_{t+1} \in \mathbb{F}_{32}$.
2. Compute $\zeta_t := \langle z_t, \mathbf{A}_1 \rangle_{\mathbb{F}_2} \oplus \langle z_{t+1}, \mathbf{A}_2 \rangle_{\mathbb{F}_2}$.
3. Copy the value ζ_t into the output register.
4. Uncompute Step 1-2.

Since \mathbf{A}_1 and \mathbf{A}_2 are predetermined constants, Step 2 can be executed by only applying CNOT gates. Hence, the T -depth of Step 2 is zero, and the above simulation requires $2 \times 64 = 2^7$ depth and the same amount of queries to $O_{BSNOW2.0}$.

Using this simulation, QFCA2 recovers the initial state of the LFSR of SNOW 2.0 (and breaks its qBPRF security) with time complexity at most $2^7 \cdot (2^{18} \cdot (512)^4 \cdot (2^{14.411})^2) \leq 2^{89.3}$, making quantum queries at most $2^7 \cdot (2^{15} \cdot (512) \cdot (2^{14.411})^2) \leq 2^{59.3}$ times.

On the other hand, the running time of the exhaustive key search with the Grover search is at least $2^{10} \cdot 2^{\kappa/2}$ for κ -bit keys, because of the following reasons: The Grover search performs $2^{\kappa/2}$ iterations to search for a κ -bit secret key. It evaluates a Boolean function f such that $f(\mathbf{x}) = 1$ iff \mathbf{x} matches the secret key in each iteration. The only way (that we are aware of) to implement such f is to compute a keystream segment for each input \mathbf{x} and check whether it matches the real keystream segment. As the initialization phase requires 32 state updates and each update involves at least one 32-bit modular addition (in the finite state machine), the T -depth of f should be at least $32 \cdot 32 = 2^{10}$.

In particular, when the key length is 256, our attack (time complexity $2^{89.3}$) is significantly faster than the generic attack by the Grover search (time complexity at least 2^{138}).

A Note on Key Recovery. Our primary aim here is to break the qBPRF security of SNOW 2.0. Still, once the LFSR's initial state is recovered, the remaining 64-bit state of the finite state machine can also be recovered with at most about

⁸ Strictly speaking, the last bit ζ_N cannot be computed due to the upper limit of the index i that can be queried to $O_{BSNOW2.0}$. So, we just set $\zeta_N := 0$. Since N is quite large ($N = 2^{512} - 1$), this modification has little effect on the attack complexity and the success probability.

2^{64} classical operations. In particular, since the initialization phase of SNOW 2.0 is reversible, we can recover the secret key with almost the same complexity.

Remark 5. A previous work [24] shows that a quantum guess-and-determine attack on SNOW 2.0 with 256-bit keys breaks the cipher in time around 2^{88} . However, the attack uses a large quantum computer of size around 2^{88} to run a parallelized Grover search, whereas our paper does not consider parallel computation. In addition, [24] defines the unit of time (resp., size) as the time to execute the target cipher once (resp., the size to implement the target cipher). Under this cost metric, if a quantum computer of size 2^{88} is available, the generic attack (simple parallelized Grover search) recovers a 256-bit key with time complexity about $\sqrt{2^{256}/2^{88}} = 2^{84}$. Thus, the attack on SNOW 2.0 with 256-bit keys is slower than the generic attack.

We also applied QFCA2 on SNOW 3G [31] and Sosemanuk [5], of which the structures are quite close to SNOW 2.0. For SNOW 3G, the time and query complexity are $2^{102.9}$ and $2^{72.9}$, which is slower than the Grover search but significantly faster than the classical attacks [35–37, 65, 80]. On Sosemanuk, the time and query complexity are $2^{101.11}$ and $2^{73.15}$. This is slower than the quantum guess-and-determine attack [24], but faster than the Grover search for long keys (e.g., 256-bit keys). See Section G of the full version of this paper [42] for details.

6.4 Discussions

We also tried speeding up other classical fast correlation attacks [35, 54, 70, 71, 75, 76, 78, 85], which recover the initial state (or even the secret key) of Grain v1 [41], Grain-128 [40], Grain-128a [1], Fruit-v2 [34], Fruit-80 [33], Plantlet [60], SNOW-V [29], and SNOW-Vi [30] faster than the classical exhaustive key search. However, we have not found Q2 attacks faster than the exhaustive key search with Grover's algorithm.

Except for SNOW-V/Vi, the problem is that the absolute correlations are too small. For SNOW-V/Vi, which uses 512-bit LFSRs and 256-bit keys, the absolute correlations are in a moderate order ($> 2^{-50}$), but still the time complexity of QFCA2 becomes at least 2^{150} due to the factor $2^{18}\ell^4$ in Eq. (22) with $\ell = 512$.

Acknowledgements. We thank anonymous reviewers for their insightful comments.

References

1. Ågren, M., Hell, M., Johansson, T., Meier, W.: Grain-128a: a new version of grain-128 with optional authentication. *Int. J. Wirel. Mob. Comput.* **5**(1), 48–59 (2011)
2. Ågren, M., Löndahl, C., Hell, M., Johansson, T.: A survey on fast correlation attacks. *Cryptogr. Commun.* **4**(3-4), 173–202 (2012)
3. Aumasson, J., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of latin dances: Analysis of salsa, chacha, and rumba. In: *FSE 2008, Revised Selected Papers*. LNCS, vol. 5086, pp. 470–488. Springer (2008)

4. Bera, D., Tharmashastha, S.: Quantum and randomised algorithms for non-linearity estimation. *ACM Transactions on Quantum Computing* **2**(2) (June 2021)
5. Berbain, C., Billet, O., Canteaut, A., Courtois, N.T., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Minier, M., Pornin, T., Sibert, H.: Sosemanuk, a fast software-oriented stream cipher. In: *New Stream Cipher Designs - The eSTREAM Finalists*, LNCS, vol. 4986, pp. 98–118. Springer (2008)
6. Berbain, C., Gilbert, H.: On the security of IV dependent stream ciphers. In: Biryukov, A. (ed.) *FSE 2007, Revised Selected Papers*. LNCS, vol. 4593, pp. 254–273. Springer (2007)
7. Berbain, C., Gilbert, H., Maximov, A.: Cryptanalysis of grain. In: Robshaw, M.J.B. (ed.) *FSE 2006, Revised Selected Papers*. LNCS, vol. 4047, pp. 15–29. Springer (2006)
8. Bernstein, D.J.: ChaCha, a variant of Salsa20. In: *Workshop Record of SASC*. vol. 8 (2008)
9. Bernstein, D.J.: The Salsa20 family of stream ciphers. In: *New Stream Cipher Designs - The eSTREAM Finalists*, LNCS, vol. 4986, pp. 84–97. Springer (2008)
10. Boneh, D., Zhandry, M.: Quantum-secure message authentication codes. In: *EUROCRYPT 2013, Proceedings*. LNCS, vol. 7881, pp. 592–608. Springer (2013)
11. Bonnetain, X., Hosoyamada, A., Naya-Plasencia, M., Sasaki, Y., Schrottenloher, A.: Quantum attacks without superposition queries: The offline simon’s algorithm. In: Galbraith, S.D., Moriai, S. (eds.) *ASIACRYPT 2019, Proceedings, Part I*. LNCS, vol. 11921, pp. 552–583. Springer (2019)
12. Bonnetain, X., Hosoyamada, A., Naya-Plasencia, M., Sasaki, Y., Schrottenloher, A.: Quantum attacks without superposition queries: The offline simon’s algorithm. In: *ASIACRYPT 2019, Part I*. LNCS, vol. 11921, pp. 552–583. Springer (2019)
13. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: On quantum slide attacks. In: Paterson, K.G., Stebila, D. (eds.) *SAC 2019, Revised Selected Papers*. LNCS, vol. 11959, pp. 492–519. Springer (2019)
14. Bonnetain, X., Schrottenloher, A., Sibleyras, F.: Beyond quadratic speedups in quantum attacks on symmetric schemes. In: Dunkelman, O., Dziembowski, S. (eds.) *EUROCRYPT 2022, Proceedings, Part III*. LNCS, vol. 13277, pp. 315–344. Springer (2022)
15. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics* **46**(4-5), 493–505 (1998)
16. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. *Contemporary Mathematics* **305**, 53–74 (2002)
17. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: *LATIN 1998*. LNCS, vol. 1380, pp. 163–169. Springer (1998)
18. Canteaut, A., Trabbia, M.: Improved fast correlation attacks using parity-check equations of weight 4 and 5. In: *EUROCRYPT 2000, Proceeding*. LNCS, vol. 1807, pp. 573–588. Springer (2000)
19. Canteaut, A.: LFSR-based stream ciphers, <https://www.rocq.inria.fr/secret/Anne.Canteaut/MPRI/chapter3.pdf> (Accessed on September 19, 2024)
20. Chepyzhov, V.V., Johansson, T., Smeets, B.J.M.: A simple algorithm for fast correlation attacks on stream ciphers. In: *FSE 2000, Proceedings*. LNCS, vol. 1978, pp. 181–195. Springer (2000)
21. Chepyzhov, V.V., Smeets, B.J.M.: On A fast correlation attack on certain stream ciphers. In: *EUROCRYPT ’91, Proceedings*. LNCS, vol. 547, pp. 176–185. Springer (1991)

22. Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: An algorithmic point of view. In: EUROCRYPT 2002, Proceedings. LNCS, vol. 2332, pp. 209–221. Springer (2002)
23. Collard, B., Standaert, F., Quisquater, J.: Improving the time complexity of matsui’s linear cryptanalysis. In: Nam, K., Rhee, G. (eds.) ICISC 2007, Proceedings. LNCS, vol. 4817, pp. 77–88. Springer (2007)
24. Ding, L., Wu, Z., Zhang, G., Shi, T.: Quantum guess and determine attack on stream ciphers. *Comput. J.* **67**(1), 292–303 (2024)
25. Dong, X., Sun, S., Shi, D., Gao, F., Wang, X., Hu, L.: Quantum collision attacks on aes-like hashing with low quantum random access memories. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 727–757. Springer (2020)
26. ECRYPT: eSTREAM: ECRYPT stream cipher project, <https://www.ecrypt.eu.org/stream/>
27. Einsele, S., Wunder, G.: Quantum speed-up of fast correlation attacks against stream ciphers. *Crypto day matters* 36
28. Ekdahl, P., Johansson, T.: A new version of the stream cipher SNOW. In: SAC 2002, Revised Papers. LNCS, vol. 2595, pp. 47–61. Springer (2002)
29. Ekdahl, P., Johansson, T., Maximov, A., Yang, J.: A new SNOW stream cipher called SNOW-V. *IACR Trans. Symmetric Cryptol.* **2019**(3), 1–42 (2019)
30. Ekdahl, P., Maximov, A., Johansson, T., Yang, J.: Snow-vi: an extreme performance variant of SNOW-V for lower grade cpus. In: WiSec 2021. pp. 261–272. ACM (2021)
31. ETSI/SAGE: Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW 3G Specification. Version 1.1 (2006)
32. Funabiki, Y., Todo, Y., Isobe, T., Morii, M.: Several milp-aided attacks against SNOW 2.0. In: CANS 2018, Proceedings. LNCS, vol. 11124, pp. 394–413. Springer (2018)
33. Ghafari, V.A., Hu, H.: Fruit-80: A secure ultra-lightweight stream cipher for constrained environments. *Entropy* **20**(3), 180 (2018)
34. Ghafari, V.A., Hu, H., Chen, Y.: Fruit-v2: Ultra-lightweight stream cipher with shorter internal state. *IACR Cryptology ePrint Archive* 2016/355 (2016)
35. Gong, X., Hao, Y., Wang, Q.: Combining milp modeling with algebraic bias evaluation for linear mask search: improved fast correlation attacks on snow. *Des. Codes Cryptogr.* **92**, 1663–1728 (2024)
36. Gong, X., Zhang, B.: Fast computation of linear approximation over certain composition functions and applications to SNOW 2.0 and SNOW 3g. *Des. Codes Cryptogr.* **88**(11), 2407–2431 (2020)
37. Gong, X., Zhang, B.: Comparing large-unit and bitwise linear approximations of SNOW 2.0 and SNOW 3g and related attacks. *IACR Trans. Symmetric Cryptol.* **2021**(2), 71–103 (2021)
38. Grassi, L., Naya-Plasencia, M., Schrottenloher, A.: Quantum algorithms for the k-xor problem. In: ASIACRYPT 2018, Proceedings, Part I. LNCS, vol. 11272, pp. 527–559. Springer (2018)
39. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: ACM STOC 1996. pp. 212–219. ACM (1996)
40. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: Proceedings 2006 IEEE International Symposium on Information Theory, ISIT 2006, The Westin Seattle, Seattle, Washington, USA, July 9-14, 2006. pp. 1614–1618. IEEE (2006)

41. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain family of stream ciphers. In: *New Stream Cipher Designs - The eSTREAM Finalists*, LNCS, vol. 4986, pp. 179–190. Springer (2008)
42. Hosoyamada, A.: Quantum algorithms for fast correlation attacks on lfsr-based stream ciphers. IACR Cryptology ePrint Archive 2024/894 (Full version of this paper)
43. Hosoyamada, A., Sasaki, Y.: Quantum Demirci-Selçuk Meet-in-the-Middle Attacks: Applications to 6-Round Generic Feistel Constructions. In: *SCN 2018*. LNCS, vol. 11035, pp. 386–403. Springer (2018)
44. Hosoyamada, A., Sasaki, Y.: Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In: *Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II*. LNCS, vol. 12106, pp. 249–279. Springer (2020)
45. ISO/IEC: 18033-4:2011 Information technology — Security techniques — Encryption algorithms. Part 4 Stream Ciphers (2011)
46. Johansson, T., Jönsson, F.: Fast correlation attacks based on turbo code techniques. In: *CRYPTO '99, Proceedings*. LNCS, vol. 1666, pp. 181–197. Springer (1999)
47. Johansson, T., Jönsson, F.: Improved fast correlation attacks on stream ciphers via convolutional codes. In: *EUROCRYPT '99, Proceeding*. LNCS, vol. 1592, pp. 347–362. Springer (1999)
48. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: *CRYPTO 2016, Part II*. LNCS, vol. 11693, pp. 207–237. Springer (2016)
49. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. *IACR Trans. Symmetric Cryptol.* **2016**(1), 71–94 (2016)
50. Kuwakado, H., Morii, M.: Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In: *ISIT 2010*. pp. 2682–2685. IEEE (2010)
51. Kuwakado, H., Morii, M.: Security on the quantum-type Even-Mansour cipher. In: *ISITA 2012*. pp. 312–316. IEEE (2012)
52. Leander, G., May, A.: Grover Meets Simon - Quantumly Attacking the FX-construction. In: *ASIACRYPT 2017*. LNCS, vol. 10625, pp. 161–178. Springer (2017)
53. Lee, J., Lee, D.H., Park, S.: Cryptanalysis of sosemanuk and SNOW 2.0 using linear masks. In: *Pieprzyk, J. (ed.) ASIACRYPT 2008, Proceedings*. LNCS, vol. 5350, pp. 524–538. Springer (2008)
54. Ma, S., Jin, C., Shi, Z., Cui, T., Guan, J.: Correlation attacks on snow-v-like stream ciphers based on a heuristic milp model. *IEEE Transactions on Information Theory, Early Access* (2023)
55. Matsui, M.: Linear cryptanalysis method for DES cipher. In: *Helleseht, T. (ed.) EUROCRYPT 1993, Proceedings*. LNCS, vol. 765, pp. 386–397. Springer (1993)
56. Meier, W.: Fast correlation attacks: Methods and countermeasures. In: *FSE 2011, Revised Selected Papers*. LNCS, vol. 6733, pp. 55–67. Springer (2011)
57. Meier, W., Staffelbach, O.: Fast correlation attacks on stream ciphers (extended abstract). In: *EUROCRYPT '88, Proceedings*. LNCS, vol. 330, pp. 301–314. Springer (1988)
58. Mihajljevic, M.J., Fossorier, M.P.C., Imai, H.: Fast correlation attack algorithm with list decoding and an application. In: *FSE 2001, Revised Papers*. LNCS, vol. 2355, pp. 196–210. Springer (2001)

59. Mihaljevic, M.J., Golic, J.D.: A fast iterative algorithm for A shift register initial state reconstruction given the noisy output sequence. In: AUSCRYPT '90, Proceedings. LNCS, vol. 453, pp. 165–175. Springer (1990)
60. Mikhalev, V., Armknecht, F., Müller, C.: On ciphers that continuously access the non-volatile key. *IACR Trans. Symmetric Cryptol.* **2016**(2), 52–79 (2016)
61. Mitzenmacher, M., Upfal, E.: Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis (2nd edition). Cambridge university press (2017)
62. National Institute of Standards and Technology: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf>
63. Naya-Plasencia, M., Schrottenloher, A.: Optimal merging in quantum k-xor and k-xor-sum algorithms. In: EUROCRYPT 2020, Proceedings, Part II. LNCS, vol. 12106, pp. 311–340. Springer (2020)
64. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press (2010)
65. Nyberg, K., Wallén, J.: Improved linear distinguishers for SNOW 2.0. In: Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4047, pp. 144–162. Springer (2006)
66. Sanders, Y.R., Low, G.H., Schere, A., Berry, D.W.: Black-box quantum state preparation without arithmetic. *Phys. Rev. Lett.* **122**, 020502 (Jan 2019)
67. Santoli, T., Schaffner, C.: Using simon’s algorithm to attack symmetric-key cryptographic primitives. *Quantum Inf. Comput.* **17**(1&2), 65–78 (2017)
68. Schrottenloher, A.: Improved quantum algorithms for the k-xor problem. In: SAC 2021, Revised Selected Papers. LNCS, vol. 13203, pp. 311–331. Springer (2021)
69. Schrottenloher, A.: Quantum linear key-recovery attacks using the QFT. In: CRYPTO 2023, Proceedings, Part V. LNCS, vol. 14085, pp. 258–291. Springer (2023)
70. Shi, Z., Jin, C., Jin, Y.: Improved linear approximations of SNOW-V and snow-vi. *IACR Cryptology ePrint Archive 2021/1105* (2021)
71. Shi, Z., Jin, C., Zhang, J., Cui, T., Ding, L., Jin, Y.: A correlation attack on full SNOW-V and snow-vi. In: EUROCRYPT 2022, Proceedings, Part III. LNCS, vol. 13277, pp. 34–56. Springer (2022)
72. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society (1994)
73. Siegenthaler, T.: Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. Inf. Theory* **30**(5), 776–780 (1984)
74. Simon, D.R.: On the Power of Quantum Computation. In: 35th Annual Symposium on Foundations of Computer Science. pp. 116–123 (1994). <https://doi.org/10.1109/SFCS.1994.365701>
75. Todo, Y., Isobe, T., Meier, W., Aoki, K., Zhang, B.: Fast correlation attack revisited - cryptanalysis on full grain-128a, grain-128, and grain-v1. In: Crypto 2018, Proceedings, Part II. LNCS, vol. 10992, pp. 129–159. Springer (2018)
76. Todo, Y., Meier, W., Aoki, K.: On the data limitation of small-state stream ciphers: Correlation attacks on fruit-80 and plantlet. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019, Revised Selected Papers. LNCS, vol. 11959, pp. 365–392. Springer (2019)

77. Wagner, D.A.: A generalized birthday problem. In: Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. LNCS, vol. 2442, pp. 288–303. Springer (2002)
78. Wang, S., Liu, M., Lin, D., Ma, L.: On grain-like small state stream ciphers against fast correlation attacks: Cryptanalysis of plantlet, fruit-v2 and fruit-80. *Comput. J.* **66**(6), 1376–1399 (2023)
79. Watanabe, D., Biryukov, A., Cannière, C.D.: A distinguishing attack of SNOW 2.0 with linear masking method. In: SAC 2003, Revised Papers. LNCS, vol. 3006, pp. 222–233. Springer (2003)
80. Yang, J., Johansson, T., Maximov, A.: Vectorized linear approximations for attacks on SNOW 3g. *IACR Trans. Symmetric Cryptol.* **2019**(4), 249–271 (2019)
81. Zeng, K., Yang, C., Rao, T.R.N.: An improved linear syndrome algorithm in cryptanalysis with applications. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO '90, Proceedings. LNCS, vol. 537, pp. 34–47. Springer (1990)
82. Zhandry, M.: How to construct quantum random functions. In: FOCS. pp. 679–687. IEEE Computer Society (2012)
83. Zhang, B., Liu, R., Gong, X., Jiao, L.: Improved fast correlation attacks on the Sosemanuk stream cipher. *IACR Trans. Symmetric Cryptol.* **2023**(4), 83–111 (2023)
84. Zhang, B., Xu, C., Meier, W.: Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0. In: CRYPTO 2015, Proceedings, Part I. LNCS, vol. 9215, pp. 643–662. Springer (2015)
85. Zhou, Z., Feng, D., Zhang, B.: Efficient and extensive search for precise linear approximations with high correlations of full SNOW-V. *Des. Codes Cryptogr.* **90**(10), 2449–2479 (2022)

Author Index

A

Adhikary, Supriya 132

B

Battarbee, Christopher 330

Borin, Giacomo 330

Brough, Julian 330

Budroni, Alessandro 35

C

Cartor, Ryann 330

Chen, Yi 207

Chevignard, Clémence 299

Chi-Domínguez, Jesús-Javier 35

D

D'Alconzo, Giuseppe 35

Di Scala, Antonio J. 35

Dong, Xiaoyang 207

F

Feng, Xiutao 358

Fouque, Pierre-Alain 299

G

Gao, Yiming 3

Guo, Jian 207

H

Han, Kyoohyung 266

He, Binang 3

Hemmert, Tobias 330

Heninger, Nadia 330

Hosoyamada, Akinori 396

Hu, Honggang 3

J

Jana, Amit 168

Jao, David 330

K

Kahrobaei, Delaram 330

Karayalçın, Sengim 99

Karmakar, Angshuman 132

Kim, Seongkwang 266

Kim, Taechan 237

Krček, Marina 99

Kulkarni, Mukul 35

Kundu, Anup Kumar 168

Kundu, Suparna 132

L

Lee, Byeonghak 266

M

Maddison, Laura 330

Mondal, Puja 132

P

Paul, Goutam 168

Perin, Guilherme 99

Persichetti, Edoardo 330

Picek, Stjepan 99

R

Ran, Lars 66

Robinson, Angela 330

S

Samardjiska, Simona 66

Schrottenloher, André 299

Shen, Yantian 207

Shi, Haotian 358

Smith-Tone, Daniel 330

Son, Yongha 266

Steinwandt, Rainer 330

W

Wang, Anyu 207

Wang, Jinghui 3

Wang, Xiaoyun 207

Wu, Lichao 99