Leonie Simpson
Mir Ali Rezazadeh Baee (Eds.)

LNCS 13915

Information Security and Privacy

28th Australasian Conference, ACISP 2023
Brisbane, QLD, Australia, July 5–7, 2023
Proceedings

Springer

Lecture Notes in Computer Science 13915

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.

Leonie Simpson · Mir Ali Rezazadeh Baee
Editors

Information Security and Privacy

28th Australasian Conference, ACISP 2023
Brisbane, QLD, Australia, July 5–7, 2023
Proceedings

Springer

Editors

Leonie Simpson
Queensland University of Technology
Brisbane, QLD, Australia

Mir Ali Rezazadeh Baee
Queensland University of Technology
Brisbane, QLD, Australia

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-031-35485-4 ISBN 978-3-031-35486-1 (eBook)
https://doi.org/10.1007/978-3-031-35486-1

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

Preface

This volume contains the refereed papers presented at the 28th Australasian Conference on Information Security and Privacy (ACISP 2023). The conference was held on 5–7 July 2023, in Brisbane, Australia and hosted by Queensland University of Technology, who provided excellent facilities and support.

The ACISP conference has been an annual event since 1996, and brings together security researchers and practitioners from academia, industry and government organizations for presentation of current developments and challenges in the domain of information security and privacy. After several years of virtual and hybrid conferences due to the COVID pandemic restrictions, 2023 marked a return to a physical conference, with opportunities to network and socialize in addition to the formal program of presentations.

For ACISP 2023, we made use of the EasyChair submission and reviewing software. The Program Committee selected 27 research papers from the 87 submissions received, following a double-blind reviewing process. Each submission received at least three reviews, and the reviewer feedback was provided to all submitting authors. We thank all authors of submitted papers - the high quality of the submissions made the task of selecting a program difficult.

This volume contains the revised versions of the 27 accepted papers. We express our thanks to Springer for their continued support of ACISP, and for their help with the conference proceedings production.

We are grateful for the efforts of the Program Committee members and external reviewers, who applied their knowledge and expertise in reviewing submissions, participating in discussions to determine which papers would be selected and providing feedback to the authors. Our deepest thanks for your efforts. The ACISP-2023 Program Committee represents both geographic and gender diversity: members were from 18 nations, and almost 35% of the committee were female. We hope future ACISP committees will continue to progress towards gender equality.

In addition to the selected research papers, the ACISP-2023 program included four invited talks on aspects of information security and privacy practice and research. QUT's Vice President (Administration) and Registrar, Leanne Harvey, spoke on the December 2022 cyber-attack on QUT. The Cyber Security CRC Research Director, Helge Janicke, discussed security research collaboration between academia, government and industry. The historical development of communications security capabilities in government was outlined, and Lennon Chang (Deakin University) talked about cultural aspects of privacy. Details of their presentations do not appear in these proceedings. However, we thank all these speakers for sharing their insight and inspiring continuing research in the information security and privacy domains.

We acknowledge the contribution of our local organizing committee members: QUT staff and research students in the Information Security discipline, whose efforts enabled the smooth running of the conference. We make special mention of a former longstanding

QUT staff member, Ed Dawson. Ed had a long involvement with ACISP, and was involved in the planning for ACISP 2023. Sadly, he passed away earlier this year. He will be greatly missed.

July 2023

Leonie Simpson
Mir Ali Rezazadeh Baee

Organization

General Chair

Josef Pieprzyk	Commonwealth Scientific and Industrial Research Organization, Data61, Australia

Publication Chairs

Leonie Simpson	Queensland University of Technology, Australia
Mir Ali Rezazadeh Baee	Queensland University of Technology, Australia

Program Committee Chairs

Leonie Simpson	Queensland University of Technology, Australia
Mir Ali Rezazadeh Baee	Queensland University of Technology, Australia

Program Committee Members

Cristina Alcaraz	University of Malaga, Spain
Elena Andreeva	Technische Universität Wien, Austria
Man Ho Au	University of Hong Kong, Hong Kong
Shi Bai	Florida Atlantic University, USA
Harry Bartlett	Queensland University of Technology, Australia
Lejla Batina	Radboud University, The Netherlands
Rishiraj Bhattacharyya	University of Birmingham, UK
Colin Boyd	Norwegian University of Science and Technology, Norway
Rongmao Chen	National University of Defense Technology, China
Chitchanok Chuengsatiansup	University of Melbourne, Australia
Amy Corman	RMIT University, Australia
Craig Costello	Microsoft Research, USA
Hui Cui	Murdoch University, Australia
Edward Dawson	Queensland University of Technology, Australia
Josep Domingo-Ferrer	Universitat Rovira i Virgili, Spain

Rafael Dowsley	Monash University, Australia
Keita Emura	National Institute of Information and Communications Technology, Japan
Ernest Foo	Griffith University, Australia
Debin Gao	Singapore Management University, Singapore
Joanne Hall	RMIT University, Australia
Jinguang Han	Southeast University, China
Jingnan He	Institute of Information Engineering of Chinese Academy of Sciences, China
Swee-Huay Heng	Multimedia University, Malaysia
Xiaolu Hou	Slovak University of Technology, Slovakia
Qiong Huang	South China Agricultural University, China
Malika Izabachène	Cosmian, France
Zahra Jadidi	Griffith University, Australia
Angelos Keromytis	Georgia Institute of Technology, USA
Dan Kim	University of Queensland, Australia
Veronika Kuchta	Florida Atlantic University, USA
Fabien Laguillaumie	University of Montpellier, LIRMM, France
Hyung Tae Lee	Chung-Ang University, South Korea
Yannan Li	University of Wollongong, Australia
Yingjiu Li	University of Oregon, USA
Shengli Liu	Shanghai Jiao Tong University, China
Yuhong Liu	Santa Clara University, USA
Rongxing Lu	University of New Brunswick, Canada
Xianhui Lu	Institute of Information Engineering, CAS, China
Siqi Ma	University of New South Wales, Australia
Mitsuru Matsui	Mitsubishi Electric, Japan
Matthew McKague	Queensland University of Technology, Australia
Weizhi Meng	Technical University of Denmark, Denmark
Chris Mitchell	Royal Holloway, University of London, UK
Kirill Morozov	University of North Texas, USA
Khoa Nguyen	University of Wollongong, Australia
Lei Pan	Deakin University, Australia
Dimitrios Papadopoulos	Hong Kong University of Science and Technology, Hong Kong
Udaya Parampalli	University of Melbourne, Australia
Josef Pieprzyk	CSIRO/Data61, Australia
Indrakshi Ray	Colorado State University, USA
Adeline Roux-Langlois	CNRS, IRISA, France
Reihaneh Safavi-Naini	University of Calgary, Canada
Amin Sakzad	Monash University, Australia
Pierangela Samarati	Università degli Studi di Milano, Italy

Luisa Siniscalchi	Technical University of Denmark, Denmark
Daniel Slamanig	Austrian Institute of Technology, Austria
Jill Slay	University of South Australia, Australia
Willy Susilo	University of Wollongong, Australia
Vanessa Teague	Australian National University, Australia
Ding Wang	Nankai University, China
Huaxiong Wang	Nanyang Technological University, Singapore
Guomin Yang	Singapore Management University, Singapore
Yuval Yarom	University of Adelaide, Australia
Xun Yi	RMIT University, Australia
Quan Yuan	University of Tokyo, Japan
Tsz Hon Yuen	University of Hong Kong, Hong Kong

External Reviewers

Léo Ackermann
Kanwal Aslam Syed
Sepideh Avizheh
Syed Badruddoja
Anuhbab Baksi
Priyanka Dutta
Sabyasachi Dutta
Jonathan Eriksen
Rami Haffar
Preston Haffey
Pavol Helebrandt
Kai Hu
Ryoma Ito
Hansraj Jangir
Dingding Jia
Yinhao Jiang
Elena Kirshanova
Jiahao Liu
Jinyu Lu
Xingye Lu
Tran Ngo
Cong Peng
Octavio Pérez-Kempner
Simone Perriello
Lucas Prabel
Sebastian Ramacher
Krijn Reijnders
Partha Sarathi Roy
Syh-Yuan Tan
Sulani Thakshila
Monika Trimoska
Peng Wang
Kexin Xu
Yanhong Xu
Haiyang Xue
Xiao Yang
Liu Zhang
Yafei Zheng

Local Organizing Committee Chairs

Leonie Simpson	Queensland University of Technology, Australia
Mir Ali Rezazadeh Baee	Queensland University of Technology, Australia

Contents

Symmetric-Key Cryptography

Public-Key Cryptography

Post-Quantum Cryptography

Cryptographic Protocols

System Security

Symmetric-Key Cryptography

Improved Differential Cryptanalysis on SPECK Using Plaintext Structures

Zhuohui Feng[1], Ye Luo[1], Chao Wang[1], Qianqian Yang[2,3], Zhiquan Liu[1], and Ling Song[1,4](✉)

[1] College of Cyber Security, Jinan University, Guangzhou, China
songling.qs@gmail.com

[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
yangqianqian@iie.ac.cn

[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

[4] National Joint Engineering Research Center of Network Security Detection and Protection Technology, Jinan University, Guangzhou, China

Abstract. Plaintext structures are a commonly-used technique for improving differential cryptanalysis. Generally, there are two types of plaintext structures: multiple-differential structures and truncated-differential structures. Both types have been widely used in cryptanalysis of S-box-based ciphers while for SPECK, an Addition-Rotation-XOR (ARX) cipher, the truncated-differential structure has not been used so far. In this paper, we investigate the properties of modular addition and propose a method to construct truncated-differential structures for SPECK. Moreover, we show that a combination of both types of structures is also possible for SPECK. For recovering the key of SPECK, we propose dedicated algorithms and apply them to various differential distinguishers, which helps to obtain a series of improved attacks on all variants of SPECK. The results show that the combination of both structures helps to improve the data and time complexity at the same time, as in the cryptanalysis of S-box-based ciphers.

Keywords: ARX ciphers · structures · differential cryptanalysis · SPECK · symmetric cryptography

1 Introduction

Symmetric ciphers play a major role in providing confidentiality. According to the nonlinear operation used in the cipher, one popular category of symmetric ciphers is S-box-based ciphers and another is ARX ciphers that are built using only modular additions, bit rotations, and bitwise XORs. When a symmetric cipher is designed, the only way to build confidence in it is through a continuous effort to evaluate its security.

Z. Feng and Y. Luo—These authors contributed equally to this work.

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 3–24, 2023.
https://doi.org/10.1007/978-3-031-35486-1_1

There are several families of attack against symmetric ciphers. Among them, differential cryptanalysis [4] and linear cryptanalysis [19] are the two major ones. Many attacks can usually be divided into two parts: a distinguisher and a key-recovery part. Specifically, a differential distinguisher constitutes a high-probability differential in a part of a cipher, say a differential $\rho \rightarrow \delta$ over E_d as shown in Fig. 1. The key-recovery part usually involves the rounds before and after the distinguisher, i.e., E_b and E_f in Fig. 1. The idea is to guess the subkeys of E_b and E_f, and check if the differential $\rho \rightarrow \delta$ occurs with a high probability for E_d. If so, the key guess is likely correct. This paper focuses on the key-recovery part.

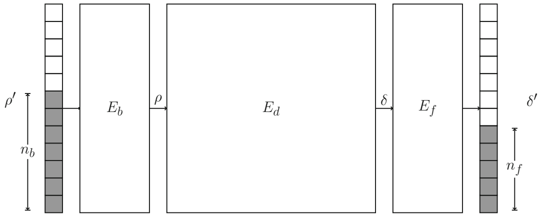


Fig. 1. Overview of differential attacks

Plaintext Structures. Along with the invention of differential cryptanalysis, structures of plaintexts are used to improve the attack. Roughly, two types of structures were proposed and widely used in the literature. Before we recall the two types, we define a ratio $\mathcal{R}_{m/p}$ between the number of messages and the number of message pairs that satisfy the input difference of the distinguisher. Then the data complexity is the product of $\mathcal{R}_{m/p}$ and the number of required pairs satisfying the input difference.

Multiple-differential structures (Type-I) This type of plaintext structures is applied to the case where the target cipher $E = E_f \circ E_d$. Without using this type of structures, a pair costs two messages, i.e., $\mathcal{R}_{m/p} = 2$. When this type of structures is used, the ratio $\mathcal{R}_{m/p}$ is expected to be lower than 2. A typical situation is that several differential trails over E_d are used simultaneously [5]. Suppose we use two trails with input differences $\varDelta_1$ and $\varDelta_2$. If we prepare N pairs of plaintexts for each input difference separately, it takes $4N$ plaintexts in total. However, when we pack them into structures, we can get 2 pairs for each input difference from a structure of 4 plaintexts, resulting in $\mathcal{R}_{m/p} = 1$. That is to say, the use of Type-I structure helps to reduce the data complexity and potentially the time complexity. Note, this type of structures was first used in the differential attack on 15-round DES [5].

Truncated-differential structures (Type-II) The second type of plaintext structures is applied to the case where the target cipher $E = E_f \circ E_d \circ E_b$ or $E = E_d \circ E_b$, i.e., there are some rounds before the distinguisher. As illustrated in Fig. 1, the input difference ρ of the distinguisher propagates backward to ρ' through E_b, where n_b bits of ρ' are active. A structure of this type consists of 2^{n_b} plaintexts where inactive bits are constant while active bits are traversed. Among each structure of 2^{n_b} plaintexts, about 2^{n_b-1} out of 2^{2n_b-1} pairs satisfy the input difference ρ of the distinguisher. In order to have N pairs of plaintexts leading to ρ, still $2N$ plaintexts are needed. That is to say, both $\mathcal{R}_{m/p}$ and the data complexity remain the same. The possible gain is to attack more rounds or reduce the time complexity. Note, this type of structures was used to reach a full-round differential attack on DES [6].

As a common technique, both types of plaintext structures have been widely used in the cryptanalysis of S-box-based ciphers, e.g., [5,7]. For ARX ciphers, Type-I structures have been used to reduce the data complexity of differential-like attacks on a series of ARX ciphers, like Chaskey [17] and SPECK [14], and Type-II structures have been used to mount impossible differential attacks and truncated differential attacks on ARX ciphers, such as XTEA, TEA and HIGHT [10,15,20]. However, to the best of our knowledge, Type-II structures have not been applied to SPECK.

In differential cryptanalysis of ARX ciphers, extending some rounds before the distinguisher is possible, but it may bring no benefit in the general case. The problem is that these ciphers typically have 32-bit or 64-bit words. Even though the input difference ρ of the distinguisher may have a few active bits, full-word subkeys of E_b have to be guessed to ensure the ρ difference. In other words, adding rounds before the distinguisher brings no benefit (compared to adding rounds after the distinguisher). Therefore, this usually does not give efficient attacks. In this paper, we study in which case the Type-II structures can be applied to SPECK and what benefits they can bring.

SPECK. SPECK is a family of ARX ciphers which were designed in 2013 by researchers from the U.S. National Security Agency (NSA) [3]. Its structure is a generalized Feistel structure and it provides excellent performance in both hardware and software. Since the design of SPECK, it has attracted intensive cryptanalysis from the community [1,8,9,11,13,18,22,24]. Besides these classical cryptanalysis, there also exist some cryptanalysis using deep learning [2,14] whose basis are differential characteristics. So far, differential-like cryptanalysis is the most powerful attack against SPECK. Note that none of the previous differential attacks employ the Type-II structures in the key-recovery attacks on SPECK.

1.1 Our Contributions

We start with studying the differential properties of modular addition. Due to the fact of modular addition that the lower bits of the output are affected only by the lower bits of the inputs, the differential propagation can be confined to the higher bits when the inputs have zero difference in the lower bits. We then introduce two parameters n_{BIL} and n_{FIL} to denote the numbers of inactive lower bits for the output of modular subtraction and addition, respectively. Based on these properties of modular addition, we formalize the construction of Type-II structures for SPECK. Moreover, we show that Type-II structures and Type-I structures can be combined for ARX ciphers and applied to SPECK.

Further, we propose three algorithms for key recovery attacks on SPECK, i.e., Algorithm A, B and C, which target the cases using Type-I structures, Type-II structures, and a combination of both, respectively. Algorithm A mainly achieves the goal of reducing the data complexity when compared with previous attacks. The aim of Algorithm B is to utilize Type-II structures so as to reduce the time complexity. Algorithm C combines the ideas of Algorithm A and B and helps to improve the data and time complexity at the same time. Note the improvement in the time complexity is proportional to n_{BIL} and n_{FIL}.

In order to prepare suitable distinguishers of SPECK for these three algorithms, we particularly search for differential trails with n_{BIL} and n_{FIL} as large as possible and show them in [12]. We then mount attacks by applying the three algorithms to newly obtained differential distinguishers. The resulting attacks on SPECK and the previous works are presented in Table 1. More detailed results on SPECK in this work are displayed in Table 8. The results show that the use of Type-II structures helps to reduce the time complexity by a factor up to 2^{15} and in many cases, both the time and data complexity are reduced to a certain extent.

Our work shows that Type-II structures are possible for SPECK and help to improve the time complexity in certain cases. Their application should not be limited to standard differential cryptanalysis of SPECK. Other attacks to which Type-II structures can be potentially applied include boomerang attacks, differential-linear attacks, impossible differential attacks, etc.

Table 1. Comparison of our attacks on SPECK with the previous works

Variants	Rounds	Probability	Data	Time	Memory	Reference
32/64	14/22	2^{-30}	2^{31}	2^{63}	2^{22}	[11]
	14/22	$2^{-29.47}$	$2^{30.47}$	$2^{62.47}$	2^{22}	[22]
	14/22	$2^{-29.37}$	$2^{31.75}$	$2^{60.99}$	$2^{41.91}$	[9]
	14/22	$2^{-27.68}$	$\mathbf{2^{30.26}}$	$\mathbf{2^{60.58}}$	2^{36}	**This**
	15/22	$2^{-30.39}$	$2^{31.39}$	$2^{63.39}$	2^{22}	[16]
	15/22	$2^{-30.39}$	$2^{31.39}$	$2^{62.25}$	-	[9]
48/72	15/22	2^{-45}	2^{46}	2^{70}	2^{22}	[13]
	15/22	$2^{-44.31}$	$2^{45.31}$	$2^{69.31}$	2^{22}	[22]
	15/22	$2^{-43.42}$	$\mathbf{2^{44.42}}$	2^{70}	2^{22}	**This**
	16/22	$2^{-46.80}$	$2^{47.80}$	$2^{71.80}$	2^{22}	[16]
	16/22	$2^{-45.78}$	$\mathbf{2^{46.78}}$	$\mathbf{2^{71.78}}$	2^{22}	**This**
48/96	16/23	2^{-45}	2^{46}	2^{94}	2^{22}	[13]
	16/23	$2^{-44.31}$	$2^{45.31}$	$2^{93.31}$	2^{22}	[22]
	16/23	$2^{-43.42}$	$\mathbf{2^{44.42}}$	2^{94}	2^{22}	**This**
	17/23	$2^{-46.80}$	$2^{47.80}$	$2^{95.80}$	2^{22}	[16]
	17/23	$2^{-45.78}$	$\mathbf{2^{46.78}}$	$\mathbf{2^{95.78}}$	2^{22}	**This**

(*continued*)

Table 1. (*continued*)

Variants	Rounds	Probability	Data	Time	Memory	Reference
64/96	19/26	2^{-62}	2^{63}	2^{95}	2^{22}	[13]
	19/26	$2^{-59.30}$	**$2^{61.88}$**	**$2^{93.34}$**	2^{68}	**This**
	19/26	$2^{-60.56}$	$2^{61.56}$	$2^{93.56}$	2^{22}	[22]
	19/26	$2^{-58.24}$	**$2^{60.82}$**	**$2^{92.28}$**	2^{68}	**This**
64/128	20/27	2^{-62}	2^{63}	2^{127}	2^{22}	[13]
	20/27	$2^{-59.30}$	**$2^{61.88}$**	**$2^{125.34}$**	2^{68}	**This**
	20/27	$2^{-60.56}$	$2^{61.56}$	$2^{125.56}$	2^{22}	[22]
	20/27	$2^{-60.73}$	$2^{63.96}$	$2^{122.69}$	$2^{77.19}$	[9]
	20/27	$2^{-58.24}$	**$2^{60.82}$**	$2^{124.28}$	2^{68}	**This**
96/96	19/28	2^{-87}	2^{88}	2^{88}	2^{22}	[13]
	19/28	2^{-86}	**2^{87}**	2^{88}	2^{22}	**This**
	20/28	$2^{-94.94}$	$2^{95.94}$	$2^{95.94}$	2^{22}	[22]
	20/28	$2^{-92.17}$	**$2^{93.17}$**	**$2^{95.75}$**	2^{22}	**This**
96/144	20/29	2^{-87}	2^{88}	2^{136}	2^{22}	[13]
	20/29	2^{-86}	**2^{87}**	2^{136}	2^{22}	**This**
	21/29	$2^{-94.94}$	$2^{95.94}$	$2^{143.94}$	2^{22}	[22]
	21/29	$2^{-92.17}$	**$2^{93.17}$**	**$2^{143.75}$**	2^{22}	**This**
	21/29	$2^{-91.03}$	**$2^{93.61}$**	**$2^{143.13}$**	2^{99}	**This**
128/128	22/32	2^{-119}	2^{120}	2^{120}	2^{22}	[13]
	22/32	2^{-117}	**2^{118}**	$2^{120.81}$	2^{22}	**This**
	23/32	$2^{-124.35}$	$2^{125.35}$	$2^{125.35}$	2^{22}	[22]
	23/32	$2^{-121.37}$	**$2^{122.37}$**	**$2^{124.95}$**	2^{22}	**This**
128/192	23/33	2^{-119}	2^{120}	2^{184}	2^{22}	[13]
	23/33	$2^{-117.19}$	**$2^{119.77}$**	**$2^{168.35}$**	2^{131}	**This**
	24/33	$2^{-124.35}$	$2^{125.35}$	$2^{189.35}$	2^{22}	[22]
	24/33	$2^{-121.37}$	**$2^{123.95}$**	**$2^{174.53}$**	2^{129}	**This**
128/256	24/34	2^{-119}	2^{120}	2^{248}	2^{22}	[13]
	24/34	$2^{-117.19}$	**$2^{119.77}$**	**$2^{232.35}$**	2^{131}	**This**
	25/34	$2^{-124.35}$	$2^{125.35}$	$2^{253.35}$	2^{22}	[22]
	25/34	$2^{-121.37}$	**$2^{123.95}$**	**$2^{238.53}$**	2^{129}	**This**

Organization. The paper is organized as follows. In Sect. 2, we introduce some preliminaries, including notations, and a description of SPECK. In Sect. 3, we study the properties of modular addition and propose some propositions, based on which Type-II structures and a combination of both types of structures can be constructed for SPECK. In Sect. 4, we introduce three key recovery algorithms for SPECK using different types of structures. We apply them to all variants of SPECK and obtain a series of improved attacks in Sect. 5. Finally, we conclude this work in Sect. 6.

2 Preliminaries

2.1 Notations

Given an n-bit word x, we denote its i^{th} bit for $i \in \{0, 1, ..., n-1\}$ by $x[i]$ and its i^{th} bit to j^{th} bit by $x[j:i]$, $i \leq j$. Given two n-bit words x and y, we denote by $x \boxplus y$ their addition modulo 2^n, by $x \boxminus y$ their modular subtraction, by $x \oplus y$ the bitwise XOR operation, by $x \vee y$ the bitwise OR operation and by $x \wedge y$ the bitwise AND operation between them. Given an n-bit word x and a positive integer i, we denote by $x \ggg i$ the n-bit word obtained by rotating x by i bits to the right, and by $x \lll i$ the word obtained by rotating x to the left.

In this paper, we denote the i^{th} round of SPECK by Round i where $i \in \{1, 2, \ldots, T\}$. The output of Round i is denoted by x_i and y_i.

2.2 Specification of SPECK

SPECK is a family of lightweight block ciphers designed by researchers from the U.S. National Security Agency (NSA) [3]. There are 10 variants, each of which is characterized by its block size $2n$ and key size mn. Some parameters for all variants of SPECK are specified in Table 2.

Table 2. The parameters of SPECK

block size $2n$	key size mn	word size n	key word size m	rot a	rot b	Rnds T
32	64	16	4	7	2	22
48	72/96	24	3/4	8	3	22/23
64	96/128	32	3/4	8	3	26/27
96	96/144	48	2/3	8	3	28/29
128	128/192/256	64	2/3/4	8	3	32/33/34

The round function of SPECK is defined as:

$$(x_{i+1}, y_{i+1}) = \mathcal{R}_{k_i}(x_i, y_i) = (((x_i \ggg a) \boxplus y_i) \oplus k_i, (y_i \lll b) \oplus ((x_i \ggg a) \boxplus y_i) \oplus k_i),$$

where k_i is the round key for $0 \leq i < T$.

The SPECK key schedule takes an initial m-word master key $(l_{m-2}, \ldots, l_0, k_0)$ and from it generates a sequence of T round key words $k_0, \ldots, k_{T-1}$ as follows:

$$l_{i+m-1} = (k_i \boxplus (l_i \ggg a)) \oplus i, \quad k_{i+1} = (k_i \lll b) \oplus l_{i+m-1}, 0 \leq i < T.$$

From the specification of SPECK, two simple properties can be obtained directly.

Property 1. If the output of Round $i+1, i \geq 0$ is known, i.e., (x_{i+1}, y_{i+1}) is known, one of the input word y_i can be determined by $y_i = (x_{i+1} \oplus y_{i+1}) \ggg b$.

Property 2. If m consecutive round keys of SPECK are known, i.e., $k_{i-m}, \cdots, k_{i-1}$ for $i > m$, we can efficiently invert the key schedule to determine $k_{i-m-1}, \cdots$, and so on until we have the original m master key words.

Consequently, the key recovery attacks of SPECK are equivalent to recovering m consecutive round keys.

3 Properties of Modular Addition and Structures

In this section, we give some properties of modular addition, based on which one could construct plaintext structures for SPECK as in the cryptanalysis of S-box-based ciphers.

3.1 Properties of Modular Addition

Given two n-bit words x and y, we let $z = x \boxplus y$ where the carry word is denoted by c. Some operation rules about the carry word c are listed as follows:

$$\begin{aligned} c[0] &= 0, \\ c[i+1] &= x[i] \wedge y[i] \oplus x[i] \wedge c[i] \oplus y[i] \wedge c[i], \quad i \in \{0, 1, \ldots, n-2\}, \\ z[i] &= x[i] \oplus y[i] \oplus c[i], \quad i \in \{0, 1, \ldots, n-1\}. \end{aligned}$$

With these in mind, we look into the XOR differences of modular addition. Let $\tilde{x} = x \oplus \alpha$, $\tilde{y} = y \oplus \beta$, $\tilde{z} = \tilde{x} \boxplus \tilde{y}$, and $\tilde{z} \oplus z = \gamma$, where α, β and γ are differences, as shown in Fig. 2. Let c, $\tilde{c}$ respectively denote the carry words of $x \boxplus y$, $\tilde{x} \boxplus \tilde{y}$ and let $\Delta c = c \oplus \tilde{c}$.

Then we have

$$\gamma[i] = z[i] \oplus \tilde{z}[i] = x[i] \oplus y[i] \oplus c[i] \oplus \tilde{x}[i] \oplus \tilde{y}[i] \oplus \tilde{c}[i] = \alpha[i] \oplus \beta[i] \oplus \Delta c[i].$$

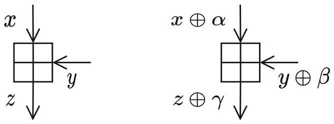


Fig. 2. The differences of the modular addition

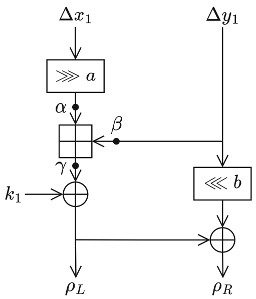


Fig. 3. The differences of Round 2

In the following, we give three propositions of the differences of modular addition.

Proposition 1. *Let $z = x \boxplus y$ and $\tilde{z} = \tilde{x} \boxplus \tilde{y}$, where $\alpha = x \oplus \tilde{x}, \beta = y \oplus \tilde{y}$ and $\gamma = z \oplus \tilde{z}$. Given β, γ where $\beta[j:0] = \gamma[j:0] = 0$, $0 \leq j < n$, then $\alpha[j:0] = 0$.*

Proof. When $\beta[0] = 0$ and $\gamma[0] = 0$, we have $\alpha[0] = \gamma[0] \oplus \beta[0] \oplus \Delta c[0] = 0$. Therefore, $\Delta c[1] = 0$. Then when $\beta[1] = 0$, $\gamma[1] = 0$, as $\alpha[1] = \gamma[1] \oplus \beta[1] \oplus \Delta c[1] = 0$, we have $\Delta c[2] = 0$, and so on. □

Definition 1 (Backward inactive lower bits (BIL). *Under the setting in Proposition 1, $\alpha[j:0] = 0$, $0 \leq j < n$. We call $\alpha[j:0]$ the backward inactive lower bits for the addition and BIL for short. Denote the length of $\alpha[j:0]$ in bits by n_{BIL}.*

Proposition 2. *Let $z = x \boxplus y$ and $\tilde{z} = \tilde{x} \boxplus \tilde{y}$, where $\alpha = x \oplus \tilde{x}, \beta = y \oplus \tilde{y}$ and $\gamma = z \oplus \tilde{z}$. Given β, γ where $\beta[j:0] = \gamma[j:0] = 0$ and $\beta[j+1] \vee \gamma[j+1] = 1$, $0 \leq j < n-1$, then $\alpha[j+1] = \beta[j+1] \oplus \gamma[j+1]$ and $\Pr[\alpha[u] = 1]$ for any $u \in (j+1, n)$ is non-zero if z and y are taken randomly (or equivalently x, y are taken randomly).*

Proof. According to Proposition 1, we have $\alpha[j:0] = 0$ and $\Delta c[j+1] = 0$. When $\beta[j+1] \vee \gamma[j+1] = 1$, we have $\alpha[j+1] = \beta[j+1] \oplus \gamma[j+1] \oplus \Delta c[j+1] = \beta[j+1] \oplus \gamma[j+1]$. We prove the subsequent cases by exhaustive calculation which is shown in Table 3(a) and 3(b).

i) For $u = j+2$, as $\beta[u-1] \vee \gamma[u-1] = 1$ and $\Delta c[u-1] = 0$, we have $\Pr[\Delta c[u] = 0] = \Pr[\Delta c[u] = 1] = \frac{1}{2}$, according to Table 3(b);

ii) For $u \geq j+3$, let $\Pr[\Delta c[u-1] = 1] = p$. According to Table 3(b),
- *a)* when $(\beta[u-1], \gamma[u-1])$=(0,0), $\Pr[\Delta c[u] = 0] = 1 - p + \frac{1}{2}p = 1 - \frac{1}{2}p$, $\Pr[\Delta c[u] = 1] = \frac{1}{2}p > 0$;
- *b)* when $(\beta[u-1], \gamma[u-1])$=(0,1), $\Pr[\Delta c[u] = 0] = \frac{1}{2}(1-p) + \frac{1}{2}p = \frac{1}{2}$, $\Pr[\Delta c[u] = 1] = \frac{1}{2}p + \frac{1}{2}(1-p) = \frac{1}{2}$;
- *c)* when $(\beta[u-1], \gamma[u-1])$=(1,0), $\Pr[\Delta c[u] = 0] = \frac{1}{2}$, $\Pr[\Delta c[u] = 1] = \frac{1}{2}$;
- *d)* when $(\beta[u-1], \gamma[u-1])$=(1,1), $\Pr[\Delta c[u] = 0] = \frac{1}{2}(1-p)$, $\Pr[\Delta c[u] = 1] = \frac{1}{2}(1-p) + p = \frac{1}{2} + \frac{1}{2}p$.

In summary, for any $u \in (j+1, n)$, $0 < \Pr[\Delta c[u] = 1] < 1$. Therefore, due to $\alpha[u] = \beta[u] \oplus \gamma[u] \oplus \Delta c[u]$, we will get $\alpha[u] = 0$ or $\alpha[u] = 1$ with certain non-zero probability. □

Similarly, on the other side we have the following proposition.

Proposition 3. *Let $z = x \boxplus y$ and $\tilde{z} = \tilde{x} \boxplus \tilde{y}$, where $\alpha = x \oplus \tilde{x}, \beta = y \oplus \tilde{y}$ and $\gamma = z \oplus \tilde{z}$. Given α, β where $\alpha[j:0] = \beta[j:0] = 0$, $0 \leq j < n$, then $\gamma[j:0] = 0$.*

Definition 2 (Forward inactive lower bits (FIL). *Under the setting of Proposition 3, $\gamma[j:0] = 0$, $0 \leq j < n$. We call $\gamma[j:0]$ the forward inactive lower bits of the addition and FIL for short. Denote the length of $\gamma[j:0]$ in bits by n_{FIL}.*

Table 3. All cases of differences of carry bits $\varDelta c[u]$ in the modular addition. (a)All cases of $\beta[u-1]$ and $\gamma[u-1]$. (b)The exhaustive calculation of $\varDelta c[u]$ for all possible $y[u-1]$, $\tilde{y}[u-1]$, $z[u-1]$, $\tilde{z}[u-1]$, $c[u-1]$, $\tilde{c}[u-1]$ where $j+1<u$.

(a)

ID	$\beta[u-1]$	$\gamma[u-1]$	$y[u-1]$,$\tilde{y}[u-1]$	$\alpha[u-1]$	
			$z[u-1]$,$\tilde{z}[u-1]$	$\varDelta c[u-1]=0$	$\varDelta c[u-1]=1$
1	0	0	0, 0, 0, 0	0	1
2			1, 1, 0, 0		
3			0, 0, 1, 1		
4			1, 1, 1, 1		
5	0	1	0, 0, 0, 1	1	0
6			0, 0, 1, 0		
7			1, 1, 0, 1		
8			1, 1, 1, 0		
9	1	0	0, 1, 0, 0	1	0
10			0, 1, 1, 1		
11			1, 0, 0, 0		
12			1, 0, 1, 1		
13	1	1	0, 1, 0, 1	0	1
14			0, 1, 1, 0		
15			1, 0, 0, 1		
16			1, 0, 1, 0		

(b)

ID	$x[u-1]$, $\tilde{x}[u-1]$, $c[u]$, $\tilde{c}[u]$, $(\varDelta c[u])$			
	$(c[u-1]$, $\tilde{c}[u-1])$ $=(0, 0)$	$(c[u-1]$, $\tilde{c}[u-1])$ $=(1, 1)$	$(c[u-1]$, $\tilde{c}[u-1])$ $=(0, 1)$	$(c[u-1]$, $\tilde{c}[u-1])$ $=(1, 0)$
1	0, 0, 0, 0 (0)	1, 1, 1, 1 (0)	0, 1, 0, 1 (1)	1, 0, 1, 0 (1)
2	1, 1, 1, 1 (0)	0, 0, 1, 1 (0)	1, 0, 1, 1 (0)	0, 1, 1, 1 (0)
3	1, 1, 0, 0 (0)	0, 0, 0, 0 (0)	1, 0, 0, 0 (0)	0, 1, 0, 0 (0)
4	0, 0, 0, 0 (0)	1, 1, 1, 1 (0)	0, 1, 0, 1 (1)	1, 0, 1, 0 (1)
5	0, 1, 0, 0 (0)	1, 0, 1, 0 (1)	0, 0, 0, 0 (0)	1, 1, 1, 0 (1)
6	1, 0, 0, 0 (0)	0, 1, 0, 1 (1)	1, 1, 0, 1 (1)	0, 0, 0, 0 (0)
7	1, 0, 1, 0 (1)	0, 1, 1, 1 (0)	1, 1, 1, 1 (0)	0, 0, 1, 0 (1)
8	0, 1, 0, 1 (1)	1, 0, 1, 1 (0)	0, 0, 0, 1 (1)	1, 1, 1, 1 (0)
9	0, 1, 0, 1 (1)	1, 0, 1, 1 (0)	0, 0, 0, 1 (1)	1, 1, 1, 1 (0)
10	1, 0, 0, 0 (0)	0, 1, 0, 1 (1)	1, 1, 0, 1 (1)	0, 0, 0, 0 (0)
11	1, 0, 1, 0 (1)	0, 1, 1, 1 (0)	1, 1, 1, 1 (0)	0, 0, 1, 0 (1)
12	0, 1, 0, 0 (0)	1, 0, 1, 0 (1)	0, 0, 0, 0 (0)	1, 1, 1, 0 (1)
13	0, 0, 0, 0 (0)	1, 1, 1, 1 (0)	0, 1, 0, 1 (1)	1, 0, 1, 0 (1)
14	1, 1, 0, 1 (1)	0, 0, 0, 1 (1)	1, 0, 0, 1 (1)	0, 1, 0, 1 (1)
15	1, 1, 1, 0 (1)	0, 0, 1, 0 (1)	1, 0, 1, 0 (1)	0, 1, 1, 0 (1)
16	0, 0, 0, 0 (0)	1, 1, 1, 1 (0)	0, 1, 0, 1 (1)	1, 0, 1, 0 (1)

3.2 Type-II Structures for SPECK

Let us come to the key recovery part of differential attacks. Given a differential $\rho \rightarrow \delta$ of SPECK, suppose we prepend one round to it. For the modular addition of this extra round, we still denote its difference propagation by $\alpha, \beta \rightarrow \gamma$, where the differences β and γ should match the input difference of the differential.

If β and γ are some random differences, α will be regarded as random as well. In terms of notations in Fig. 1, it leads to a large n_b, i.e., a large number of plaintext bits will be active. In contrast, if the lower bits or the least significant bit of β and γ are 0, we can know for sure the same number of lower bits of α are zero as well, while its higher bits or the most significant bit can be either 1 or 0. That is to say, for certain proper differentials, the number of active bits in the plaintext can be small.

A common experience in differential cryptanalysis of S-box-based ciphers is that a small n_b is desirable, so it would be interesting to see the effect of building Type-II structures on the active higher bits of some input words of SPECK.

In the following, we will show how Type-II structures of SPECK can be built by exploiting Proposition 1 and 2 and discuss the possibility of applying both types of structures to ARX ciphers simultaneously.

Type-II Structures for SPECK. Note that the first round of SPECK acts as a whitening layer, so it can be covered for free. Now we consider adding two rounds before a differential of SPECK. Suppose the input difference of the differential is $\rho = (\rho_L, \rho_R)$ and its probability is 2^{-w}. Further, suppose the input difference ρ of the differential propagates backward to (α, β) marked in Fig. 3 such that, according to Proposition 1 and 2, α has the form

$$\alpha = 0b\underbrace{*\cdots*}_{s}c\underbrace{0\cdots0}_{n-s-1},$$

where $0b$ indicates that the binary sequence is followed, the most significant s bits of α can take any possible value and c is a known constant. As for β, it can be computed from ρ and thus is fixed.

We then construct structures at the beginning of the second round. In this situation, pairs formed from the structures will satisfy the input difference ρ of the differential probabilistically rather than deterministically. We will show that the required data remains the same as that in Dinur's attacks [11].

The goal of the data collection phase is to generate pairs (x_1, y_1) and (x_1', y_1') whose difference is (α, β). From such pairs, the difference between $(x_1 \oplus k_0 \ggg a) \boxplus (y_1 \oplus k_0)$ and $(x_1' \oplus k_0 \ggg a) \boxplus (y_1' \oplus k_0)$ is expected to equal γ with probability 2^{-s} under some unknown round keys k_0. To achieve this, we could generate such pairs using twin structures:

$$\begin{aligned}S &= \{(x_1, y_1) | x_1[n-s-1:0], y_1 = c_1, x_1[n-1:n-s] \in \{0,1\}^s\},\\ S' &= S \oplus \Delta = \{(x_1', y_1') | (x_1, y_1) \in S, x_1'[n-s-1:0] = x_1[n-s-1:0] \oplus c0\cdots0,\\ &\quad y_1' = y_1 \oplus \beta, x_1'[n-1:n-s] \in \{0,1\}^s\}, \text{where } \Delta = 0b0\cdots0c0\cdots0||\beta. \qquad (1)\end{aligned}$$

From such a pair of twin structures, we can generate 2^{2s} pairs of (x_1, y_1) and (x'_1, y'_1), and 2^s pairs are expected to lead to γ, meeting the input difference of the distinguisher.

Suppose 2^t pairs of twin structures are used. We need 2^w pairs of (x_1, y_1) and (x'_1, y'_1) that satisfy the input difference of the distinguisher to have at least one right pair for the differential. Therefore, $2^{s+t} = 2^w$ and $s + t = w$. The total data complexity for 2^t pairs of twin structures is $D = 2^{t+s+1} = 2^{w+1}$, which is the same as that in Dinur's attacks. This confirms again that Type-II structures do not help to reduce the data complexity. Therefore, in this paper, the purpose of using Type-II structures is not to reduce the data complexity, but to reduce the time complexity. This will be shown in Sect. 4.2.

3.3 Combining both Types of Structures

Recall that the use of Type-I structures helps to reduce the data complexity. What if we use both types of structures simultaneously? Can we reduce the time and data complexity at the same time?

Suppose we have two differentials $(\rho_1 \to \delta)$ and $(\rho_2 \to \delta)$ of the same probability for SPECK and ρ_1 and ρ_2 propagate back over one round to (α_1, β_1) and (α_2, β_2), respectively, where α_1 and α_2 share the same number of zero bits in the lower part. Then we choose four structures as follows:

$$S, S \oplus \Delta_1, S \oplus \Delta_2, S \oplus \Delta_1 \oplus \Delta_2, \tag{2}$$

where Δ_1 and Δ_2 are derived from (α_1, β_1) and (α_2, β_2) in a way presented in Eq. (1). Actually, these four sets form a Type-I structure of Type-II structures.

If the size of each Type-II structure is 2^s, then we can get about 2×2^s pairs satisfying ρ_1 and about 2×2^s pairs satisfying ρ_2. Now the ratio between the number of messages and the number of message pairs meeting one of the input differences becomes 1, i.e., $\mathcal{R}_{m/p} = 1$. Generally, if there are h differentials, $\mathcal{R}_{m/p} = \frac{2}{h}$, which is much lower than 2. This shows that the combination of both types of structures may bring benefits of each together.

4 New Key Recovery Algorithms for SPECK

In this section, we propose three generic key recovery algorithms for SPECK which respectively use Type-I structures, Type-II structures and a combination of both types of structures. In these algorithms, we reuse the 2-round attack proposed by Dinur in [11]. In particular, we will highlight the benefit each algorithm may bring.

4.1 Algorithm a Using Type-I Structures

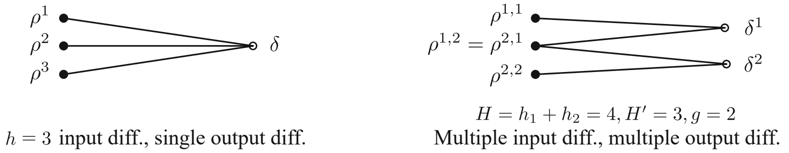

ρ^1
ρ^2
ρ^3
δ
$h = 3$ input diff., single output diff.
$\rho^{1,1}$
$\rho^{1,2} = \rho^{2,1}$
$\rho^{2,2}$
δ^1
δ^2
$H = h_1 + h_2 = 4, H' = 3, g = 2$
Multiple input diff., multiple output diff.

Fig. 4. Two examples where Type-I structures can be applied

Suppose we have a set of h differentials $\{\rho^j \to \delta\}$ for $j = 1, \cdots, h$ over r rounds and the corresponding probabilities are p^j where $\sum_{j=1}^{h} p^j = \hat{p} = 2^{-\hat{w}}$, as given in the left part of Fig. 4. Moreover, this case can also be extended to that with multiple output differences (see the right part of Fig. 4). Suppose we have g output differences δ^i, $i = 1, \cdots, g$. Each output difference δ^i corresponds to h_i input differences, i.e., $\{\rho^{i,j} \to \delta^i\}$ of probability $p^{i,j}$ for $j = 1, \cdots, h_i$. Let

$$\sum_{i,j} p^{i,j} = \hat{p} = 2^{-\hat{w}}, H = h_1 + \cdots + h_g.$$

Suppose among the H input differences $\rho^{i,j}$s, H' of them are linearly independent when we treat them as binary vectors. We can attack $1 + r + m$ rounds following the procedure below.
For $i = 1, \cdots, 2^t$:

1. Construct a structure S of $2^{H'}$ data by taking the H' independent input differences $\rho^{i,j}$ as a basis.
2. Guess the last $m - 2$ round keys. Do partial decryptions to have intermediate values (x_{r+3}, y_{r+3}).
   (a) For each input difference $\rho^{i,j}$ and each output difference δ^i, $j = 1, \cdots, h_i$, $i = 1, \cdots, g$, generate $2^{h_i - 1} \times 2^{H' - h_i}$ pairs of (P, P') meeting $\rho^{i,j}$. For each pair,
      i. Run the 2-round attack of [11] using $(\Delta x_{r+1}, \Delta y_{r+1}) = \delta$, $(\Delta x_{r+3}, \Delta y_{r+3})$ and (x_{r+3}, y_{r+3}) and return solutions of k_{r+1} and k_{r+2};
      ii. For each returned value of k_{r+1} and k_{r+2} together with the guessed $m - 2$ subkeys, recover the master key and test it using trial encryptions, and return it if the trial encryptions succeed.

It takes $2^t \times 2^{H'}$ data and let $2^t \times 2^{H'-1} \times \sum_{i,j} p^{i,j} = 1$. Therefore $H' + t = \hat{w} + 1$. Then we can summarize the complexities of Attack A as follows.

- $D_{\mathrm{A}} = 2^{\hat{w}+1}$ plaintexts;
- $T_{\mathrm{A}} = 2^t \times 2^{(m-2)n} \times H \times 2^{H'-1} \times 2 = c_{\mathrm{A}} \cdot 2^{(m-2)n+\hat{w}} \times H = c_{\mathrm{A}} \cdot 2^{(m-2)n} \frac{1}{\left(\sum_{i,j} p^{i,j}\right)/H} = c_{\mathrm{A}} \cdot 2^{(m-2)n} \cdot \frac{H}{\hat{p}}$ encryptions, where $c_{\mathrm{A}} = 2$, $m = 2, 3, 4$;
- $M_{\mathrm{A}} = \max\{2^{22}, 2^{H'}\}$.

Example. Table 4 lists some examples which demonstrate the improvement brought by Algorithm A. In these two attacks on 14-round SPECK32/64, the probability of all differential trails we use is 2^{-30}. The result shows the data complexity is reduced by using Type-I structures.

Table 4. Example of attacks using Algorithm A

Variants	Split	#Trails	h	g	Data	Time	Mem.	Ref
32/64	1+9+4	1	1	1	2^{31}	2^{63}	2^{22}	[11]
	1+9+4	5	5	1	$2^{28.68}$	2^{63}	2^{22}	Alg. A
	1+9+4	15	15	6	$2^{27.09}$	2^{63}	2^{22}	Alg. A

Experiment. We mainly test whether we can get right pairs as expected. We construct Type-I structures and use a 5-round distinguisher for the 10-round SPECK32/64 attack. The experiment results demonstrate that no matter how many right pairs we set, we can always get the expected number of right pairs.

4.2 Algorithm B Using Type-II Structures

Suppose we have a differential $\rho = (\rho_L, \rho_R) \rightarrow \delta = (\delta_L, \delta_R)$ with probability $p = 2^{-w}$. Also, suppose the output difference δ leads to n_{FIL} forward inactive lower bits for the addition in the next round and the input difference results in n_{BIL} backward inactive lower bits for the addition in the earlier round. The goal of Algorithm B is to attack $2 + r + (m - 1)$ rounds for $m = 3, 4$.

Switch to the Counting Method. When we use Type-II structures, there are two rounds before the distinguisher. If we guess the last $m - 2$ round keys, the time complexity will exceed 2^{mn}, because we need to process more pairs than that in Dinur's attack. Thus, in our attack we only guess the last $m - 3$ round keys. For Round $r + 3$ and Round $r + 4$, we can also mount the 2-round attack to recover round keys k_{r+2} and k_{r+3}. However, due to Property 2, testing the recovered round keys using trial encryptions is possible only when we have information of m consecutive round keys. In this case, we only have the keys of the last $m - 1$ consecutive rounds (or with the first round key). This makes the enumerating method infeasible.

A way to get around this issue is to switch to the common counting method. The number of message pairs that meet the input difference ρ and the output difference δ under each possible key value is recorded. The key values with the highest counters are the likeliest right key. Thus, a shortlist of round key candidates can be obtained according to the counters, which helps to give an efficient key recovery of the master key.

The Algorithm. In this algorithm, we guess the involved round key bits of k_0 which are needed for verifying the input difference ρ. As the higher bits of both k_0 and $k_0 \ggg a$ affect the verification, more than $s = n - n_{\mathsf{BIL}} - 1$ bits of k_0

are needed. In the following procedures, we guess these bits of k_0 together with k_{r+m} if $m = 4$ and make counters for other involved round keys k_{r+2}, k_{r+3}. The detailed procedures of Algorithm B are as follows.

1. Construct 2^t pairs of twin structures (S, S') in a way described in Eq. 1. Each structure has 2^s plaintext-ciphertext pairs, where $s = n - n_{\mathsf{BIL}} - 1$.
2. Guess $\min\{n - n_{\mathsf{BIL}} + a, n\}$ bits of k_0 and the full k_{r+m} if $m = 4$. Do partial encryptions and decryptions.
   (a) Initialize counters for k_{r+2} and k_{r+3}.
   (b) For each pair of twin structures:
      i. Store the data of one structure into a hash table according to the state before the key addition of the second round and $(x_{r+4} \oplus y_{r+4})[n_{\mathsf{FIL}} + b : b]$ and look up the table with data from the other structure. There will be $2^{s-n_{\mathsf{FIL}}-1}$ pairs of data meeting the input difference and the $(n_{\mathsf{FIL}} + 1)$-bit fixed difference.
      ii. For each pair, mount the 2-round attack to recover k_{r+2}, k_{r+3} and update the counters.
   (c) Select 2^ℓ (e.g., $l < n$) candidates for k_{r+2}, k_{r+3} with top counters. Guess k_{r+1} and test the correctness by trial encryptions.

If we set the number of right pairs to one, then $2^t \times 2^s \times p = 1$, $t + s = w$, and the data complexity is 2^{w+1}, which is the same as that in Dinur's attack. If we want to have a higher success probability, we can increase the data complexity. For the computation of success probability, we follow the formula in Selçuk's work [21] (see Eq. 3 of Sect. 4.3). However, for a convenient comparison of time complexities between Algorithm B and Dinur's attack, we simply let the data complexity be the same (while we may trade the data for a higher success probability in concrete applications). The memory complexity for storing data and counters is $D + 2^{2n} \approx 2^{2n}$, and the time complexity is dominated by step (b) and (c). Specifically,

- $D_{\rm B} = 2^{w+1}$ plaintexts;
- $T_{\rm B} = 2^{(m-3)n+\min\{n-n_{\mathsf{BIL}}+a,n\}+t}(2^{s+1} + 2^{s-n_{\mathsf{FIL}}-1}) + 2^{(m-2)n+\min\{n-n_{\mathsf{BIL}}+a,n\}+\ell}$ encryptions, $m = 3, 4$;
- $M_{\rm B} = 2^{2n}$ for counters.

As the number of candidate keys 2^ℓ is flexible, it is reasonable to assume the second term of $T_{\rm B}$ is not dominant. Let us focus on the first term. We can see that if n_{BIL} is greater than the rotation number a, this attack is already more efficient than Dinur's attack in terms of time complexity.

Improvement. Note that the time complexities of step (i) and (ii) are 2^{s+1} and $2^{s-n_{\mathsf{FIL}}-1}$ under each guess, respectively, which are not balanced. A strategy to balance them is to guess fewer key bits of k_0. There are three types of bits in k_0: bits used once (bits in the red lines in Fig. 5), twice (bits in the blue line in Fig. 5) and none for verifying ρ difference. Our strategy is to guess all bits that are used twice and partial bits that are used once. Let us analyze case by case:

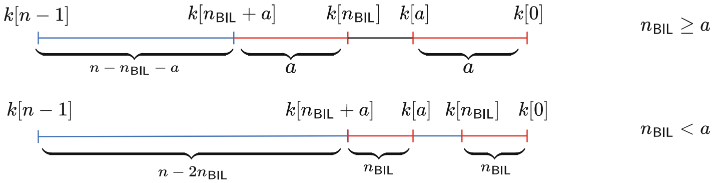


Fig. 5. Guess bits of k_0, where bits in the blue line are all guessed and y bits in the red lines are guessed. (Color figure online)

- When $n_{\mathsf{BIL}} \geq a$, $n - n_{\mathsf{BIL}} + a$ bits of k_0 are needed. If we guess $n - n_{\mathsf{BIL}} - a + y$ bits of k_0 with $0 \leq y \leq 2a$, we will get a $(n - n_{\mathsf{BIL}} - 2a + y - 1)$-bit filter, i.e., an $(s - 2a + y)$-bit filter, instead of an s-bit filter. Now the time complexity of step (ii) becomes $2^{s-n_{\mathsf{FIL}}-1+2a-y}$. Meanwhile, the number of guessed key bits of k_0 is reduced by $2a - y$ bits and the memory complexity is increased by 2^{2a-y}.
- When $n_{\mathsf{BIL}} < a$, a full k_0 is needed. If we guess $n - 2n_{\mathsf{BIL}} + y$ bits of k_0 with $0 \leq y \leq 2n_{\mathsf{BIL}}$, we will get an $(s - 2n_{\mathsf{BIL}} + y)$-bit filter, instead of an s-bit filter. Now the time complexity of step (ii) becomes $2^{s-n_{\mathsf{FIL}}-1+2n_{\mathsf{BIL}}-y}$. Meanwhile, the number of guessed key bits of k_0 is reduced by $2n_{\mathsf{BIL}} - y$ bits and the memory complexity is increased by $2^{2n_{\mathsf{BIL}}-y}$.

The advantage of Algorithm B over Dinur's attack in terms of the time complexity is summarized in Table 5, which is denoted by *ad*.

Table 5. Advantage of Algorithm B

$n_{\mathsf{BIL}} \geq a$	$2a \geq n_{\mathsf{FIL}} + 1$	$y = 2a - n_{\mathsf{FIL}} - 1$	$ad = n_{\mathsf{BIL}} - a + n_{\mathsf{FIL}} + 1$
	$2a < n_{\mathsf{FIL}} + 1$	$y = 0$	$ad = n_{\mathsf{BIL}} + a$
$n_{\mathsf{BIL}} < a$	$2n_{\mathsf{BIL}} \geq n_{\mathsf{FIL}} + 1$	$y = 2n_{\mathsf{BIL}} - n_{\mathsf{FIL}} - 1$	$ad = n_{\mathsf{FIL}} + 1$
	$2n_{\mathsf{BIL}} < n_{\mathsf{FIL}} + 1$	$y = 0$	$ad = 2n_{\mathsf{BIL}}$

Example. We take the attack on SPECK128/256 as an example, as shown in Table 6. In the attack a 19-round differential trail with probability 2^{-119} is used where $n_{\mathsf{BIL}} = 23, n_{\mathsf{FIL}} = 2$. We mount attacks on 24 rounds using Algorithm B and set the number of right pairs $\mu = 3$ and let $\ell = 55$ so that the success probability is high (about 90% in the calculation). As can be seen the time complexity is reduced by using Type-II structures and by taking $n_{\mathsf{BIL}}, n_{\mathsf{FIL}}$ into account.

Table 6. Example of attacks using Algorithm B

Variants	Split	Prob.	n_{BIL}	n_{FIL}	Data	Time	Mem.	Ref
128/256	1+19+4	2^{-119}	-	-	2^{120}	2^{248}	2^{22}	[13]
	2+19+3	2^{-119}	23	2	$2^{121.58}$	$2^{231.58}$	2^{131}	Alg. B

Experiment. Using Algorithm B, we try to mount a 13-round attack on SPECK32/64 with a 8-round distinguisher. The result shows that the correct key ranks high. We set $\mu = 2$ and the highest counter is 6. Specifically, among all possible keys, 16 values take this count, including the correct key.

4.3 Algorithm C Using Type-I and Type-II Structures

Suppose we have g output differences δ^i, $i = 1, \cdots, g$, all of which lead to at least n_{FIL} forward inactive lower bits for the addition in the next round. Each output difference δ^i corresponds to h_i input differences, i.e., $\rho^{i,j} \rightarrow \delta^i$ of probability $p^{i,j}$ for $j = 1, \cdots, h_i$. Suppose all these input differences result in at least n_{BIL} backward inactive lower bits for the addition in the earlier round. Let

$$\sum_{i,j} p^{i,j} = \hat{p} = 2^{-\hat{w}}, H = h_1 + \cdots + h_g.$$

Suppose among the H input differences $\rho^{i,j}$s, H' of them are linearly independent. The goal is to attack $2 + r + (m - 1)$ rounds for $m = 3, 4$.

The Algorithm. This algorithm is a combination of Algorithm A and B.

1. Construct 2^t Type-I structures of $2^{H'}$ Type-II structures and get the corresponding ciphertexts. Each type-II structure has 2^s plaintexts, where $s = n - n_{\mathsf{BIL}} - 1$.
2. If $n_{\mathsf{BIL}} \geq a$, guess $n - n_{\mathsf{BIL}} - a + y$ bits of k_0 with $0 \leq y \leq 2a$; otherwise, guess $n - 2n_{\mathsf{BIL}} + y$ bits of k_0 with $0 \leq y \leq 2n_{\mathsf{BIL}}$. Guess the full k_{r+m} if $m = 4$. Do partial encryptions and decryptions.
   (a) Initialize counters for k_{r+2}, k_{r+3}, and $2 \times \min\{a, n_{\mathsf{BIL}}\} - y$ bits of k_0.
   (b) For each Type-I structure, each input difference $\rho^{i,j}$ and each output difference δ^i, $i = 1, \cdots, g$, $j = 1, \cdots, h_i$, there are $2^{h_i - 1} \times 2^{H' - h_i}$ pairs of twin structures.
      i. For each pair of twin structures, store the data of one structure into a hash table according to the state before the key addition of the second round and $(x_{r+m} \oplus y_{r+m})[n_{\mathsf{FIL}} + b : b]$ and look up the table with data from the other structure. Then, when $n_{\mathsf{BIL}} \geq a$ (resp. $n_{\mathsf{BIL}} < a$) there will be $2^{s - n_{\mathsf{FIL}} - 1 + 2a - y}$ (resp. $2^{s - n_{\mathsf{FIL}} - 1 + 2n_{\mathsf{BIL}} - y}$) pairs of data partly meeting the input difference and the $(n_{\mathsf{FIL}} + 1)$-bit fixed difference.
      ii. For each pair of data, mount the 2-round attack to recover k_{r+2}, k_{r+3}; recover other involved bits of k_0 by looking up a precomputed table[1]

[1] This precomputed table takes a small memory of $2^{3 \times (2 \times \min\{a, n_{\mathsf{BIL}}\} - y)}$.

which is created in a way described in the improvement strategy of Sect. 4.2. Update the counters accordingly.

(c) Select 2^ℓ (e.g., $\ell < n$) candidates with top counters. Guess k_{r+1} and test the correctness by trial encryptions.

It takes $2^t \times 2^{H'} \times 2^s$ data. For a convenient comparison of time complexities between Algorithm C and Dinur's attack, we simply set the number of right pairs to one. Then we have $2^t \times 2^{H'-1} \times 2^s \times \sum_{i,j} p^{i,j} = 1$ and thus $H' + t + s = \hat{w} + 1$. However, we may trade the data for a higher success probability in attacks on concrete applications. Assume n_{FIL} is much smaller than s, which holds in general. Then we can summarize the complexities of Algorithm C as follows.

- $D_{\mathrm{C}} = 2^{\hat{w}+1}$ plaintexts;
- When $n_{\mathsf{BIL}} \geq a$, $T_{\mathrm{C}} = 2^{(m-3)n+n-n_{\mathsf{BIL}}-a+y} \times (2^t \times H \times 2^{H'-1} \times (2^{s+1} + 2^{s-n_{\mathsf{FIL}}-1+2a-y}) + 2^{n+\ell}))$ encryptions for $m = 3, 4$.
  - If $2a \geq n_{\mathsf{FIL}} + 1$, $T_{\mathrm{C}} = 2^{(m-2)n-(n_{\mathsf{BIL}}-a+n_{\mathsf{FIL}}+1)} \times (\frac{H}{\hat{p}} \times 2 + 2^{n+\ell})$. $M_{\mathrm{C}} = 2^{2n+n_{\mathsf{FIL}}+1}$ for counters.
  - If $2a < n_{\mathsf{FIL}} + 1$, $T_{\mathrm{C}} = 2^{(m-2)n-(n_{\mathsf{BIL}}+a)} \times (\frac{H}{\hat{p}} \times 2 + 2^{n+\ell})$. $M_{\mathrm{C}} = 2^{2n+2a}$ for counters.
- When $n_{\mathsf{BIL}} < a$, $T_{\mathrm{C}} = 2^{(m-3)n+n-2n_{\mathsf{BIL}}+y} \times (2^t \times H \times 2^{H'-1} \times (2^{s+1} + 2^{s-n_{\mathsf{FIL}}-1+2n_{\mathsf{BIL}}-y}) + 2^{n+\ell}))$ encryptions for $m = 3, 4$.
  - If $2n_{\mathsf{BIL}} \geq n_{\mathsf{FIL}} + 1$, $T_{\mathrm{C}} = 2^{(m-2)n-(n_{\mathsf{FIL}}+1)} \times (\frac{H}{\hat{p}} \times 2 + 2^{n+\ell})$. $M_{\mathrm{C}} = 2^{2n+n_{\mathsf{FIL}}+1}$ for counters.
  - If $2n_{\mathsf{BIL}} < n_{\mathsf{FIL}} + 1$, $T_{\mathrm{C}} = 2^{(m-2)n-2n_{\mathsf{BIL}}} \times (\frac{H}{\hat{p}} \times 2 + 2^{n+\ell})$. $M_{\mathrm{C}} = 2^{2n+2n_{\mathsf{BIL}}}$ for counters.

As Algorithm A, the data complexity is reduced. The total advantage of this attack in time complexity over Dinur's attack is the same as shown in Algorithm B of Sect. 4.2.

Data and the Probability of Success. We can calculate the success probability using the formula in Selçuk's work,

$$P_s = \Phi\left(\frac{\sqrt{\mu S_N} - \Phi^{-1}(1 - 2^{-(n_k-\ell)})}{\sqrt{S_N + 1}}\right), \tag{3}$$

where μ is the number of right pairs, S_N is the signal-to-noise ratio, n_k is the number of key bits involved in the key recovery phase, and 2^ℓ is the size of the short list of the key candidates.

Notably, to have a competitive probability of success, having one right pair in the counting based algorithm is usually not enough. Therefore, in concrete attack we increase the data complexity by a factor μ to $\mu \times 2^{\hat{w}+1}$. We typically set $\mu = 3$ to have a reasonable success probability. In turn, this increases the data complexity, probably leading to a data complexity higher than that of the Dinur's attack.

Example. We take SPECK32/64 as an example. The best 9-round differential trails of SPECK32 have a probability 2^{-30}. Among them, a few have $n_{\mathsf{BIL}} = 3$,

$n_{\mathsf{FIL}} = 3$. Using Algorithm C, we set $\mu = 3$ and let $\ell = 14$. When 3 trails with total probability $2^{-28.42}$ are used, the success probability of attack is about 66%. When two more trails are added, the total probability of all 5 differential trails is $2^{-27.68}$, and the success probability is about 76%. The results of these attacks are listed in Table 7.

Table 7. Example of attacks using Algorithm C

Variants	Split	#Trail(s)	n_{BIL}	n_{FIL}	Data	Time	Mem.	Ref
32/64	1+9+4	1	-	-	2^{31}	2^{63}	2^{22}	[11]
	2+9+3	3	3	3	2^{31}	$2^{60.58}$	2^{36}	Alg. C
	2+9+3	5	2	3	$2^{30.26}$	$2^{60.58}$	2^{36}	Alg. C

Experiment. We mount a 10-round attack on SPECK32/64 using Algorithm C. In this experiment, we use four trails, the probability of which are all 2^{-13}. When we provide 2^{14} message pairs, i.e., $\mu = 2$, whose differences match the input difference of the distinguisher, the results show that the highest counter is 5 and the correct key does take this count. Among all possible keys, there are roughly 2^{12} candidates whose counters are more than or equal to 2.

4.4 Discussions and Extensions

In this paper, we find a way to construct Type-II structures for SPECK. It allows to add one more round before the differential distinguisher and leads to better attacks in certain situations. From the application to SPECK, we can see that the benefit of Type-II structures is weakened by the nonlinearity of the key schedule and the rotation $\ggg a$ in the round function. Thus, it is expected that the benefit of Type-II structures may vary from cipher to cipher.

Limitation of Algorithm B and C. The algorithms using Type-II structures can be applied to SPECK variants where $m = 3, 4$. When $m = 2$, the number of pairs to be processed 2^{t+2s} may exceed 2^{mn} before guessing any round key. Note this is usually the case since $t + s$ is close to $2n$ in most attacks. Therefore, our new attack is difficult to apply to the variants of SPECK with $m = 2$, namely, when the key size equals the block size.

Extensions to Variants of Differential Attacks. Besides the standard differential cryptanalysis, the new technique for constructing Type-II structures is potentially useful in various variants of differential attacks on ARX ciphers. Such variants include boomerang/rectangle attacks, differential-linear attacks, impossible differential attacks, etc.

5 Applications to SPECK

In this section, we apply the three new algorithms proposed in Sect. 4 to key recovery attacks of SPECK, where we reuse the SAT-based method [23] to search for suitable differentials or differential trails which is briefly described in [12].

5.1 Summary of Results

In this subsection, we mount key recovery attacks on SPECK by applying the new algorithms in Sect. 4 using new differential distinguishers which are shown in detail in [12].

The results are summarized in Table 8 where we label each distinguisher with an ID whose details are provided in [12], and the comparison to the related works is presented in Table 1.

Table 8. All improved attacks using our new Algorithm

Variants	Split	ID	Prob.	$n_{\mathsf{BIL}}, n_{\mathsf{FIL}}$	Data	Time	Mem.	Method
32/64	2+8+3	21	$2^{-21.68}$	3,3	$2^{24.26}$	$2^{55.58}$	2^{36}	Alg. C
	1+9+4	1	$2^{-26.09}$	-	$2^{27.09}$	2^{63}	2^{22}	Alg. A
	2+9+3	22	$2^{-27.68}$	2,3	$2^{30.26}$	$2^{60.58}$	2^{36}	Alg. C
48/72	1+11+3	2	$2^{-43.42}$	-	$2^{44.42}$	2^{70}	2^{22}	Alg. A
	2+11+2	23	$2^{-45.42}$	11,7	2^{48}	2^{62}	2^{56}	Alg. C
	1+12+3	3	$2^{-45.78}$	-	$2^{46.78}$	$2^{71.78}$	2^{22}	Alg. A
48/96	1+11+4	2	$2^{-43.42}$	-	$2^{44.42}$	2^{94}	2^{22}	Alg. A
	2+11+3	23	$2^{-45.42}$	11,7	2^{48}	2^{86}	2^{56}	Alg. C
	1+12+4	3	$2^{-45.78}$	-	$2^{46.78}$	$2^{95.78}$	2^{22}	Alg. A
64/96	1+15+3	4	2^{-60}	-	2^{61}	2^{95}	2^{22}	Alg. A
	1+15+3	5	$2^{-58.91}$	-	$2^{59.91}$	$2^{93.91}$	2^{22}	Alg. A
	2+15+2	15	$2^{-60.53}$	5,2	$2^{63.11}$	$2^{92.11}$	2^{67}	Alg. B
	2+15+2	25	$2^{-59.30}$	2,3	$2^{61.88}$	$2^{93.34}$	2^{68}	Alg. C
	2+15+2	26	$2^{-58.24}$	2,3	$2^{60.82}$	$2^{92.28}$	2^{68}	Alg. C
64/128	1+15+4	4	2^{-60}	-	2^{61}	2^{127}	2^{22}	Alg. A
	1+15+4	5	$2^{-58.91}$	-	$2^{59.91}$	$2^{125.91}$	2^{22}	Alg. A
	2+15+3	15	$2^{-60.53}$	5,2	$2^{63.11}$	$2^{124.11}$	2^{67}	Alg. B
	2+15+3	25	$2^{-59.30}$	2,3	$2^{61.88}$	$2^{125.34}$	2^{68}	Alg. C
	2+15+3	26	$2^{-58.24}$	2,3	$2^{60.82}$	$2^{124.28}$	2^{68}	Alg. C
96/96	1+16+2	6	2^{-86}	-	2^{87}	2^{88}	2^{22}	Alg. A
	1+17+2	7	$2^{-92.17}$	-	$2^{93.17}$	$2^{95.75}$	2^{22}	Alg. A
96/144	1+16+3	6	2^{-86}	-	2^{87}	2^{136}	2^{22}	Alg. A
	2+16+2	16	2^{-87}	7,0	$2^{89.58}$	$2^{136.58}$	2^{97}	Alg. B
	2+16+2	17	2^{-92}	15,0	$2^{94.58}$	$2^{134.58}$	2^{97}	Alg. B
	2+16+2	27	$2^{-85.19}$	7,0	$2^{87.77}$	$2^{137.35}$	2^{97}	Alg. C
	1+17+3	7	$2^{-92.17}$	-	$2^{93.17}$	$2^{143.75}$	2^{22}	Alg. A
	2+17+2	28	$2^{-91.03}$	7,2	$2^{93.61}$	$2^{143.13}$	2^{99}	Alg. C
128/128	1+19+2	8	2^{-117}	-	2^{118}	$2^{120.81}$	2^{22}	Alg. A
	1+20+2	9	$2^{-121.37}$	-	$2^{122.37}$	$2^{124.95}$	2^{22}	Alg. A
128/192	1+19+3	8	2^{-117}	-	2^{118}	$2^{184.81}$	2^{22}	Alg. A
	2+19+2	19	2^{-119}	23,2	$2^{121.58}$	$2^{167.58}$	2^{131}	Alg. B
	2+19+2	29	$2^{-117.19}$	23,2	$2^{119.77}$	$2^{168.35}$	2^{131}	Alg. C
	1+20+3	9	$2^{-121.37}$	-	$2^{122.37}$	$2^{188.95}$	2^{22}	Alg. A
	2+20+2	20	$2^{-123.17}$	23,0	$2^{125.75}$	$2^{173.75}$	2^{129}	Alg. B
	2+20+2	30	$2^{-121.37}$	23,0	$2^{123.95}$	$2^{174.53}$	2^{129}	Alg. C
128/256	1+19+4	8	2^{-117}	-	2^{118}	$2^{248.81}$	2^{22}	Alg. A
	2+19+3	19	2^{-119}	23,2	$2^{121.58}$	$2^{231.58}$	2^{131}	Alg. B
	2+19+3	29	$2^{-117.19}$	23,2	$2^{119.77}$	$2^{232.35}$	2^{131}	Alg. C
	1+20+4	9	$2^{-121.37}$	-	$2^{122.37}$	$2^{252.95}$	2^{22}	Alg. A
	2+20+3	20	$2^{-123.17}$	23,0	$2^{125.75}$	$2^{237.75}$	2^{129}	Alg. B
	2+20+3	30	$2^{-121.37}$	23,0	$2^{123.95}$	$2^{238.53}$	2^{129}	Alg. C

Now we highlight some features of the results shown in Table 8. Algorithm A is used to attack all variants of SPECK while Algorithm B and C are applied to variants with $m = 3, 4$. We use proper differentials for reduced SPECK with the largest number of rounds that can be attacked and for SPECK reduced to fewer rounds, we use differential trails. When we use Algorithm B and C which are based on the counting method, in order to have a reasonable success probability, we set the number of right pairs to a larger value than the one used in the enumerating method of Dinur's attack and Algorithm A. This increases the data complexity and the time complexity by a factor. For distinguishers with small signal-to-noise ratio, e.g., the ones for SPECK32 and SPECK48, there is no advantage if we use Algorithm B. As a result, Algorithm A and C, which uses multiple trails or differentials, give better results in most cases.

Note that the results of Algorithm A achieve the goal of reducing the data complexity when compared with previous attacks. The aims of using Algorithm B are to make use of Type-II structure and reduce the time complexity. Algorithm C combines the ideas of Algorithm A and B and helps to improve the data and time complexity at the same time. However, the memory complexity of the attacks using Algorithm B and C might be higher than the previous attacks which use Dinur's algorithm.

6 Conclusion

In this paper, we study the properties of modular addition and find a method to construct Type-II structures for SPECK. We also show that a combination of both types of structures is possible for ARX ciphers. To demonstrate the effect of these structures, we apply them to SPECK and obtain a series of improved attacks on all variants of SPECK. Our results confirm that Type-II structures help to reduce the time complexity and the combination of both types of structures helps to improve the data and time complexity at the same time, as in the cryptanalysis of S-box-based ciphers. Besides the standard differential cryptanalysis, we believe Type-II structures can be potentially applied to other differential-like cryptanalysis for ARX ciphers, such boomerang attacks, differential-linear attacks, impossible attacks, etc.

Acknowledgement. The authors would like to thank anonymous reviewers for their helpful comments and suggestions. The work of this paper was supported by the National Natural Science Foundation of China (Grants 62022036, 62132008, 62202460).

References

1. Abed, F., List, E., Lucks, S., Wenzel, J.: Differential cryptanalysis of round-reduced Simon and Speck. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 525–545. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46706-0_27

2. Bao, Z., Guo, J., Liu, M., Ma, L., Tu, Y.: Enhancing differential-neural cryptanalysis. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. LNCS, vol. 13791, pp. 318–347. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-22963-3_11
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive (2013)
4. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_1
5. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991). https://doi.org/10.1007/BF00630563
6. Biham, E., Shamir, A.: Differential cryptanalysis of the full 16-round DES. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 487–496. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_34
7. Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_1
8. Biryukov, A., Roy, A., Velichkov, V.: Differential analysis of block ciphers SIMON and SPECK. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 546–570. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46706-0_28
9. Biryukov, A., dos Santos, L.C., Teh, J.S., Udovenko, A., Velichkov, V.: Meet-in-the-filter and dynamic counting with applications to SPECK. Cryptology ePrint Archive (2022)
10. Chen, J., Wang, M., Preneel, B.: Impossible differential cryptanalysis of the lightweight block ciphers TEA, XTEA and HIGHT. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 117–137. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31410-0_8
11. Dinur, I.: Improved differential cryptanalysis of round-reduced SPECK. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 147–164. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13051-4_9
12. Feng, Z., Luo, Y., Wang, C., Yang, Q., Liu, Z., Song, L.: Improved differential cryptanalysis on SPECK using plaintext structures. Cryptology ePrint Archive (2023)
13. Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: MILP-based automatic search algorithms for differential and linear trails for SPECK. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 268–288. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_14
14. Gohr, A.: Improving attacks on round-reduced SPECK32/64 using deep learning. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 150–179. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_6
15. Hong, S., Hong, D., Ko, Y., Chang, D., Lee, W., Lee, S.: Differential cryptanalysis of TEA and XTEA. In: Lim, J.-I., Lee, D.-H. (eds.) ICISC 2003. LNCS, vol. 2971, pp. 402–417. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24691-6_30
16. Lee, H., Kim, S., Kang, H., Hong, D., Sung, J., Hong, S.: Calculating the approximate probability of differentials for ARX-based cipher using SAT solver. J Korea Inst. Inf. Secur. Cryptol. **28**(1), 15–24 (2018)
17. Leurent, G.: Improved differential-linear cryptanalysis of 7-round Chaskey with partitioning. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol.

9665, pp. 344–371. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_14

18. Liu, Z., Li, Y., Jiao, L., Wang, M.: A new method for searching optimal differential and linear trails in ARX ciphers. IEEE Trans. Inf. Theory **67**(2), 1054–1068 (2020)
19. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 81–91. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-47555-9_7
20. Moon, D., Hwang, K., Lee, W., Lee, S., Lim, J.: Impossible differential cryptanalysis of reduced round XTEA and TEA. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 49–60. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45661-9_4
21. Selçuk, A.A.: On probability of success in linear and differential cryptanalysis. J. Cryptol. **21**(1), 131–147 (2008)
22. Song, L., Huang, Z., Yang, Q.: Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In: Liu, J.K., Steinfeld, R. (eds.) ACISP 2016. LNCS, vol. 9723, pp. 379–394. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40367-0_24
23. Sun, L., Wang, W., Wang, M.: Accelerating the search of differential and linear characteristics with the SAT method. IACR Trans. Symmetric Cryptol. 269–315 (2021)
24. Wang, F., Wang, G.: Improved differential-linear attack with application to round-reduced SPECK32/64. In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. LNCS, pp. 792–808. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-09234-3_39

Linear Cryptanalysis and Its Variants with Fast Fourier Transformation Technique on MPC/FHE/ZK-Friendly $\mathbb{F}_p$-Based Ciphers

Zeyu Xu[1,2], Shiyao Chen[3], Meiqin Wang[1,2,4](✉), and Puwen Wei[1,2,4]

[1] Key Laboratory of Cryptologic Technology and Information Security,
Ministry of Education, Shandong University, Jinan, China
xuzeyu@mail.sdu.edu.cn, {mqwang,pwei}@sdu.edu.cn

[2] School of Cyber Science and Technology, Shandong University, Qingdao, China

[3] Strategic Centre for Research in Privacy-Preserving Technologies and Systems,
Nanyang Technological University, Singapore, Singapore
shiyao.chen@ntu.edu.sg

[4] Quan Cheng Shandong Laboratory, Jinan, China

Abstract. The emergence of advanced cryptographic protocols has promoted the developments of many applications, such as secure multi-party computation (MPC). For this reason, new symmetric-key primitives have been designed to natively support the finite field $\mathbb{F}_p$ with odd characteristic for better efficiencies. However, some well-studied symmetric cryptanalytic methods and techniques over $\mathbb{F}_2^n$ cannot be applied to these new primitives over $\mathbb{F}_p$ directly. Considering less standard design approaches adopted in these novel MPC-friendly ciphers, these proposals are in urgent need of full investigations; generalizations of the traditional cryptanalytic tools and techniques to $\mathbb{F}_p$ will also contribute to better understand the security of these new designs.

In this paper, we first show that the Fast Fourier Transform (FFT) technique for the estimations of correlation, introduced by Collard *et al.* at ICISC 2007, can be applied to $\mathbb{F}_p$ and significantly improves the complexity of Matsui's *Algorithm 2* over $\mathbb{F}_p$. Then, we formalize the differential-linear (DL) cryptanalysis to $\mathbb{F}_p$. Inspired by the differential-linear connectivity table (DLCT) introduced by Bar-On *et al.* at EUROCRYPT 2019, we also include the DLCT into the consideration, and find the relation between DLCT and differential distribution table (DDT) over $\mathbb{F}_p$. Finally, we mount key recovery attacks on a version of HADESMiMC, which is a SHARK-like MPC-friendly block cipher proposed by Grassi *et al.* at EUROCRYPT 2020. We denote this version as HADESMiMC-128 in this paper. For linear cryptanalysis with the FFT technique, we can attack 7 rounds of HADESMiMC-128. For DL cryptanalysis, a 7-round key recovery attack of HADESMiMC-128 is also mounted but with better time and data complexity. It should be noted that the attacks are still far from threatening the security of the full 14-round HADESMiMC-128.

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 25–52, 2023.
https://doi.org/10.1007/978-3-031-35486-1_2

Keywords: MPC-Friendly Block Ciphers · Linear Cryptanalysis · Differential-Linear Cryptanalysis · Fast Fourier Transformation · HADES

1 Introduction

Secure multi-party computation, fully homomorphic encryption (FHE) and zero-knowledge (ZK) proof system have been one of the most productive lines of research in recent years. In the context of these advanced applications, symmetric-key primitives are still needed, but different efficiency metrics, such as the multiplicative complexity and depth, are taken into consideration. This is because these MPC/FHE/ZK systems can be described by arithmetic operations, which are defined over finite fields $\mathbb{F}_p$ of odd characteristic, so if one chooses AES [22] as the underlying primitive for these protocols, the arithmetic conversion from $\mathbb{F}_2$ to $\mathbb{F}_p$ is usually expensive and becomes the bottleneck. Naturally, many dedicated symmetric-key primitives such as [2–4,28–30] are designed to work natively over $\mathbb{F}_p$ for better performances. Meanwhile, these innovative constructions and new operating fields with odd characteristic also pose some potential threats to these primitives, from algebraic attacks [1,25] to statistical attacks, e.g. differential and integral attacks [9]. However, linear cryptanalysis and its variants have not got enough attention for these MPC/FHE/ZK-friendly ciphers, only several of them carefully evaluated the resistance against linear attacks, such as Ciminion [24] and Reinforced Concrete [29]. As is well known, linear cryptanalysis [35] introduced by Matsui in 1993 has been one of the most classical cryptanalytic methods for symmetric primitives. In [35], Matsui proposed the partial key-recovery attack known as *Algorithm 2* in the form of a last round attack. Later, Collard *et al.* [19] found that *Algorithm 2* could be accelerated by using the FFT technique. Recently, by using linear cryptanalysis with the dedicated FFT acceleration, Flórez-Gutiérrez *et al.* [26,27] improved the attacked rounds and firstly presented 28-/29-round linear key recovery attacks on PRESENT, which has 31 total rounds and is one of the ISO/IEC standard [16] for lightweight block ciphers. Besides linear cryptanalysis combined with advanced techniques, different kinds of variants of linear cryptanalysis have also been developed, including differential-linear cryptanalysis [32], multiple linear [12] and zero-correlation linear cryptanalysis [17], to name a few. From all these developments, linear cryptanalysis and its related techniques or variants have been shown to be very powerful tools for design or cryptanalysis of symmetric ciphers over $\mathbb{F}_2^n$. Naturally, generalizing linear cryptanalysis to $\mathbb{F}_p$ is an important task, especially for this very new designing direction that also needs more in-depth cryptanalysis. In this vein, thanks to Baignères *et al.* [5] and Bleichenbacher [13], more accurate estimations of linear correlation over $\mathbb{F}_p$ are provided. However, linear cryptanalysis with advanced techniques and its variants are still in pressing need to be transformed from $\mathbb{F}_2^n$ to $\mathbb{F}_p$, which are expected to be basic tools at hand for cryptographers, and facilitate the design and analysis of these newly proposed MPC/FHE/ZK-friendly primitives.

Our Contributions. In this paper, focusing on the prime field $\mathbb{F}_p$ with odd characteristic, we study the Fast Fourier Transform technique for linear cryptanalysis and generalize the differential-linear cryptanalysis.

FFT Technique for Linear Cryptanalysis over $\mathbb{F}_p$. In order to (partially) accelerate the computation of key recovery attacks for linear cryptanalysis over the prime field, we first extend the general framework of Matsui's *Algorithm 2* given by Biryukov *et al.* [12] to $\mathbb{F}_p$. Then, we prove that the FFT-based optimization can be also integrated into *Algorithm 2* over $\mathbb{F}_p$ to gain significant improvement in terms of the complexity. In a nutshell, the $p^k \times p^k$ matrix D, used to store the correlations for all p^k guessed keys in Matsui's *Algorithm 2*, has the same underlying structure of a *level circulant matrix* [23], which can be further diagonalized by two k-dimensional Discrete Fourier Transform matrices F^*, F as

$$D = F^* diag(\lambda_D) F.$$

Finally, the time complexity of Matsui's *Algorithm 2* over $\mathbb{F}_p$ can be significantly reduced from $\mathcal{O}(p^{2k})$ to $\mathcal{O}(kp^{k+1})$.

DL Cryptanalysis and Generating DLCT with FFT over $\mathbb{F}_p$. For the first time, we formalize the DL cryptanalysis over $\mathbb{F}_p$, including the DLCT first introduced by Bar-On *et al.* [6] for ciphers over $\mathbb{F}_2^n$. Then the similar convenient transformation from DDT to DLCT is also revealed over $\mathbb{F}_p$; to be specific, assuming a DLCT of size $p^t \times p^t$, the time complexity of fast generating the DLCT from DDT with FFT can be greatly reduced from $\mathcal{O}(p^{3t})$ to $\mathcal{O}(tp^{2t+1})$.

Applications of Linear Cryptanalysis with FFT Technique and DL Cryptanalysis over $\mathbb{F}_p$. We apply our generalized cryptanalytic tools and techniques to a version of HADESMiMC, which is an MPC/FHE/ZK-friendly block cipher belonging to HADES family proposed by Grassi *et al.* [30]. This version, which we denote as HADESMiMC-128, has 14 rounds with 5 full S-box layers at the head/tail and 4 partial S-box layers in the middle, denoted by $14 = (5 + 4 + 5)$. When estimating the security of statistical attacks, designers only consider the full S-box layer and not partial S-box layer. They state that no statistical attacks can attack more than $4 = (2 + 0 + 2)$ rounds. Our following attacks are based on partial S-box round, that is, $6 = (1 + 4 + 1)$ and $7 = (1 + 4 + 2)$. To mount key recovery attacks, we first find a 6-round linear approximation and construct a 6-round DL distinguisher, which are the current longest linear/DL distinguishers of HADESMiMC-128. Then on the one hand, we demonstrate that the linear attack can cover one more round when using the FFT technique. On the other hand, a 7-round DL attack is performed with better data and time complexity than the 7-round linear attack. The results of our attacks are given in Table 1.

The designers of HADESMiMC have defined two security levels, one is $\lceil \log_2 p \rceil \times t$ corresponding to the traditional symmetric scheme, and the other is $\lceil \log_2 p \rceil$ for MPC applications. We emphasize that our attack is on the security level of the former, and cannot affect the application of the strategy in MPC. In addition, the experiment in this paper uses 8-bit p instead of large prime numbers. This is due to the high complexity of the operations in the large prime number field, which makes the search of the distinguisher more difficult. We emphasize that the main work of this paper is to reveal the basic theory for all p, however, the search technique for large p is incomplete, so we do not propose corresponding experimental results.

Table 1. Summary of key recovery attacks on HADESMiMC-128.

Prime p	Attack	Rounds	Data	Time*
Any	Statistical [30]	4		
251	Linear	6	$2^{117.9}$ KP	$2^{121.98}$
251	Linear + FFT	7	$2^{123.9}$ KP	$2^{124.72}$
251	Differential-Linear	7	$2^{110.63}$ CP	$2^{115.53}$

* Evaluated by encryption units (6 or 7 rounds).
KP: Known plaintext.
CP: Chosen plaintext.

Orgnization of the Paper. In Sect. 2, we review the differential and linear cryptanalysis over $\mathbb{F}_p$ and describe the HADESMiMC block cipher. In Sect. 3, we give a new application of the FFT-based technique in linear cryptanalysis over $\mathbb{F}_p$. In Sect. 4, we adapt differential-linear cryptanalysis to $\mathbb{F}_p$. Two key recovery attacks of 7-round HADESMiMC-128 are presented in Sect. 5.

2 Preliminaries

Some notations used throughout the paper are given as below.

- lower case letters(*e.g.* u, v, x): denote vectors in $\mathbb{F}_p^t$;
- $|\cdot|$: denotes the modulus of a complex number;
- $u[i]$ or u_i: denotes the i-th word of a vector $u \in \mathbb{F}_p^t$, where $u = (u_{t-1}, u_{t-2}, \cdots, u_0)$.
- $u \cdot v$: denotes the inner product operation, $u \cdot v = \sum_{i=0}^{t-1} u[i] \times v[i] \ mod \ p$;
- $EX(h(x))$: denotes the expectation of function $h(x)$, where the independent variable x is uniformly distributed in the domain.

2.1 Differential and Linear Cryptanalysis over $\mathbb{F}_p$

Differential [11] and linear cryptanalysis [35] are the most widely used symmetric cryptanalytic methods. For linear cryptanalysis, Baignères *et al.* [5] have developed the correlation analysis of symmetric-key primitives designed over abelian groups, which has been used to evaluate the security of Ciminion [24] against linear attacks. The core idea is that a character is an additive homomorphism from $\mathbb{F}_p^t$ into $S_p = \{\omega \in \mathbb{C} : \omega^p = 1\}$, and any character is of the form

$$\chi_u(x) = e^{\frac{2\pi\sqrt{-1}}{p} u \cdot x},$$

for some $u \in \mathbb{F}_p^t$. Now, let $f : \mathbb{F}_p^t \to \mathbb{F}_p$, then its correlation [5] can be defined as

$$cor(f) = \frac{1}{p^t} \sum_{x \in \mathbb{F}_p^t} e^{\frac{2\pi\sqrt{-1}}{p} f(x)}.$$

When $f(x)$ is replaced by a linear approximation, it has the following definition.

Definition 1 (Correlation over $\mathbb{F}_p$ [5]). *Given a function $F : \mathbb{F}_p^t \to \mathbb{F}_p^t$, for a linear mask pair (u, v) where $u, v \in \mathbb{F}_p^t$, let $f(x) = v \cdot F(x) - u \cdot x$, then the correlation of the linear approximation (u, v) of F is defined as*

$$cor_F(u, v) = \frac{1}{p^t} \sum_{x \in \mathbb{F}_p^t} \overline{\chi_u}(x)\chi_v(F(x)) = \frac{1}{p^t} \sum_{x \in \mathbb{F}_p^t} e^{\frac{2\pi\sqrt{-1}}{p}(v \cdot F(x) - u \cdot x)} = EX(e^{\frac{2\pi\sqrt{-1}}{p} f(x)}).$$

Definition 2 (Linear Probability over $\mathbb{F}_p$ [5]). *The linear probability of a function F is*

$$LP_F(u, v) = |cor_F(u, v)|^2.$$

Linear Probability of Linear Trail over $\mathbb{F}_p$. In [5], the linear probability of a r-round linear trail $\Omega = (u^0, u^1, \cdots, u^r)$ is given as

$$LP(\Omega) = \prod_{i=1}^{r} LP(u^{i-1}, u^i).$$

For multipath, they also obtain the linear hull effect.

Theorem 1 (Linear Hull Effect over $\mathbb{F}_p$ [5]). *Let $\Omega = (u^0, u^1, \cdots, u^r)$ be all r-round linear trail with input/output masks u^0 and u^r, then we have*

$$LP(u^0, u^r) = \sum_{u^1, u^2 \cdots, u^{r-1}} LP(\Omega).$$

Probability of Success. According to the *Theorem 7* in [5], suppose that the amount of data required for a key recovery attack using a trail with linear probability LP is N, and $\alpha = \frac{N}{2} \times LP$. Then the probability of success of finding the correct key in one attack is $P_s = 1 - e^{-\frac{\alpha}{2}}$, and the keys need a further exhaustive search accounts for $e^{-\frac{\alpha}{2}}$ of all guessed keys.

In the rest of the paper, we will use α as a parameter when estimating the data complexity and success probability.

The differential probability over $\mathbb{F}_2^n$ can be easily adapted to the prime field $\mathbb{F}_p$.

Definition 3 (Differential Probability over $\mathbb{F}_p$). *Given a function $F : \mathbb{F}_p^t \to \mathbb{F}_p^t$, $(\Delta_{in}, \Delta_{out})$ is a differential pair for the difference of modular subtraction, where $\Delta_{in}, \Delta_{out} \in \mathbb{F}_p^t$. Its differential probability is defined as*

$$DP_F[\Delta_{in} \to \Delta_{out}] = \frac{\#\{x \in \mathbb{F}_p^t | F(x) - F(x - \Delta_{in}) = \Delta_{out}\}}{p^t}.$$

3 FFT Technique for Linear Cryptanalysis over $\mathbb{F}_p$

In this section, we introduce and generalize the FFT technique for linear cryptanalysis over $\mathbb{F}_p$. We begin with the adaption of Matsui's *Algorithm 2* to the prime field $\mathbb{F}_p$. Then, we dive into the optimizations of the time complexity of the correlation estimations, which is inspired by Collard *et al.* 's work [19].

3.1 General Framework of Matsui's *Algorithm 2* over $\mathbb{F}_p$

Following the framework in [12], we first adapt Matsui's *Algorithm 2* to the prime field $\mathbb{F}_p$. Suppose a linear distinguisher of a cipher E over $\mathbb{F}_p^t$ requires N pairs of plaintext and ciphertext for the key recovery attack. Let input and output masks of this distinguisher be Γ_{in} and Γ_{out} respectively, and there are k $(1 \leq k < t)$ words of the secret key over $\mathbb{F}_p^t$ that need to be guessed. Note that Matsui's *Algorithm 2* only considers one-round key recovery attack for simplicity and can be divided into two phases as below.

Distillation Phase

- Initialize an array A of size p^k.
- For each plaintext-ciphertext pair (P, C), extract k words $(C[i_k], \cdots, C[i_1])$ of $C \in \mathbb{F}_p^t$ for k active S-boxes, where $0 \leq i_1 < \cdots < i_k < t$. Then use $C' = C[i_k]||\cdots||C[i_1]$ as the index of A and update $A[C'] = A[C'] + e^{-\frac{2\pi\sqrt{-1}}{p}(\Gamma_{in}\cdot P)}$.

Analysis Phase

- Initialize an array COR of size p^k.
- Extract k words $(\Gamma_{out}[i_k], \cdots, \Gamma_{out}[i_1])$ of $\Gamma_{out} \in \mathbb{F}_p^t$ for k active S-boxes and denote $\Gamma'_{out} = \Gamma_{out}[i_k]||\cdots||\Gamma_{out}[i_1]$, where $0 \leq i_1 < i_2 < \cdots < i_k < t$.
- Build a table D of size $p^k \times p^k$. For each guessed key $K' \in \mathbb{F}_p^k$ and each ciphertext $C \in \mathbb{F}_p^t$, we obtain $C' \in \mathbb{F}_p^k$ from C. Then the entry of D is updated as below

$$D[K', C'] = e^{\frac{2\pi\sqrt{-1}}{p}(\Gamma'_{out}\cdot S_k^{-1}(C'-K'))},$$

  where S_k^{-1} represents the inverse of the last one-round for k active S-boxes.
- For each guessed key $K' \in \mathbb{F}_p^k$, the experimental correlation can be evaluated as

$$COR[K'] = \sum_{C'} D[K', C'] \times A[C'].$$

Considering the complexity of these two phases, the major cost is the generations of the table D and COR. D will be accessed p^k times for each guessed K', which leads to the time complexity of $\mathcal{O}(p^k \times p^k)$. D will be computed row by row, and each row multiplied with A to generate COR, which leads to the time complexity of $\mathcal{O}(p^k \times p^k)$.

3.2 FFT for Correlation Estimation over $\mathbb{F}_p$

In order to reduce the time complexity of the analysis phase discussed above, it is natural to explore some optimizations for the computation of the table D and its matrix multiplication. Inspired by the FFT technique introduced by Collard *et al.* [19] for linear cryptanalysis over $\mathbb{F}_2^n$, we now dedicate to study properties of the structure of the table D and show that the FFT technique also can be used in the case of a prime field $\mathbb{F}_p$ with odd characteristic.

Firstly, we find that the structure of the table D, also a square matrix, is equivalent to the structure of $p^k \times p^k$ matrix Q with $Q[i,j] = j - i$, where $i = (i_{k-1}, \cdots, i_0) \in \mathbb{F}_p^k$, $j = (j_{k-1}, \cdots, j_0) \in \mathbb{F}_p^k$ and $j - i = (j_{k-1} - i_{k-1}, \cdots, j_0 - i_0)$. That is, $D[i,j] = e^{\frac{2\pi\sqrt{-1}}{p}(\Gamma'_{out} \cdot S_k^{-1}(Q[i,j]))}$. Therefore, we can assert that when a method can decompose Q, it can also decompose D. We then introduce some definitions and properties of the circulant matrix [23] as follows, which helps us reveal the underlying structure of Q further.

Definition 4 (circulant [23]). *A circulant matrix is a square matrix where the elements of each row are identical to those of the previous row, but are moved one position to the right and wrapped around.*

Definition 5 (m-block circulant [23]). *Let $B_1, B_2, \cdots, B_m$ be square matrices each of order n, a m-block circulant circulant matrix with each block of size $n \times n$ is an $mn \times mn$ matrix of the form*

$$\begin{pmatrix} B_1 & B_2 & \cdots & B_m \\ B_m & B_1 & \cdots & B_{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ B_2 & B_3 & \cdots & B_1 \end{pmatrix}.$$

Definition 6 (level circulant [23])

- *A matrix is level-1 circulant with type(n) iff it is circulant of size $n \times n$.*
- *A matrix is level-2 circulant with type(m, n) iff it is m-block circulant matrix and each block is a level-1 circulant with type(n).*
- *A matrix is level-q circulant with type($m_q, m_{q-1}, \cdots, m_1$) iff it is m_q-block circulant matrix and each block is a level-$(q-1)$ circulant with type($m_{q-1}, \cdots, m_1$).*

Proposition 1. *The matrix Q of order p^k is a level-k circulant with type $\underbrace{(p, p, \cdots, p)}_{k\ times}$.*

Proof. Consider a simple induction on k. For $k = 1$, Q is as below

$$\begin{pmatrix} 0 & 1 & \cdots & p-1 \\ p-1 & 0 & \cdots & p-2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & \cdots & 0 \end{pmatrix},$$

and the claim is obvious. For $k > 1$, assume the claim holds for $k-1$, and express Q as a block matrix of order p

$$Q = \begin{pmatrix} Q_{0,0} & Q_{0,1} & \cdots & Q_{0,p-1} \\ Q_{1,0} & Q_{1,1} & \cdots & Q_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{p-1,0} & Q_{p-1,1} & \cdots & Q_{p-1,p-1} \end{pmatrix}.$$

Each block of Q is a square matrix of size p^{k-1}, then for any $a, b \in \mathbb{F}_p$ and $i, j \in \mathbb{F}_p^{k-1}$, we have

$$Q_{a,b}[i,j] = (b, j_{k-2}, \cdots, j_1) - (a, i_{k-2}, \cdots, i_1) = (b-a, j-i).$$

Thus for any $d \in \mathbb{F}_p$, we have

$$Q_{a+d,b+d}[i,j] = ((b-d)-(a-d), j-i) = (b-a, j-i) = Q_{a,b}[i,j],$$

which means that Q is a block circulant with order p. As by the induction hypothesis, $Q_{0,0}$ is a level-$(k-1)$ circulant with type $\underbrace{(p, \cdots, p)}_{k-1\ times}$. Furthermore, we have $Q_{a,b}[i,j] = (b-a, j-i)$ and $Q_{0,0}[i,j] = (0, j-i)$, which means that each $Q_{a,b}$ has the same structure as $Q_{0,0}$. And thus, all blocks of Q are level-$(k-1)$ circulant with type $\underbrace{(p, \cdots, p)}_{k-1\ times}$, Q is a level-k circulant with type $\underbrace{(p, p, \cdots, p)}_{k\ times}$. This completes the proof.

These level circulant matrices can be decomposed by the following conclusion.

Theorem 2 **([23]).** *A level-q circulant Q with type $(m_q, m_{q-1}, \cdots, m_1)$ is diagonalizable by the unitary matrix $F = F_{m_q} \otimes F_{m_{q-1}} \otimes \cdots \otimes F_{m_1}$ as below*

$$Q = F^* diag(\lambda_Q) F,$$

where λ_Q is the vector of eigenvalues of Q, the symbol $\otimes$ is the Kronecker product which is defined in Appendix B and F_{m_l}, $1 \le l \le q$ is the Fourier matrix of size $m_l \times m_l$ defined by

$$F_{m_l}[i,j] = \frac{1}{\sqrt{m_l}} \omega^{ij} \quad (0 \le i, j \le m_l - 1),$$

with $\omega = e^{\frac{2\pi\sqrt{-1}}{m_l}}$. F^* is the conjugate transpose of F, which satisfies $FF^* = I$.

F and F^* are the k-dimensional Discrete Fourier Transform (DFT) matrices, and the multidimensional FFT allows us to quickly compute the matrix-vector product with F and F^*. The complexity of computing this product decreases from $\mathcal{O}(m_q^2 m_{q-1}^2 \cdots m_1^2)$ to $\mathcal{O}(m_q m_{q-1} \cdots m_1 (m_q + m_{q-1} + \cdots + m_1))$ [21]. As shown in [19], $diag(\lambda_Q)$ can be obtained by a Fourier transform

$$\lambda_Q = FQ[:1]\sqrt{m_q m_{q-1} \cdots m_1},$$

where $Q[:1]$ is a vector defined by the first column of Q. Since D and Q have exactly the same structure, the above conclusion is also applicable to D, just replace $Q[:1]$ and λ_Q with $D[:1]$ and λ_D.

For the analysis phase, we only need to calculate all the values of the first column of D with the complexity of p^k, and then we can further complete the matrix vector multiplication of D and A. That is

$$DA = F^* diag(\lambda_D) FA = p^{\frac{k}{2}} F^* diag(FD[:,1]) FA$$

which uses the FFT technique three times as below.

- Firstly, computing an FFT on A;
- Secondly, computing an FFT to obtain $diag(\lambda_D)$ then multiplying the resulting vector by $diag(\lambda_D)$ can be basically ignored;
- Thirdly, performing an FFT on the resulting vector by F^*.

Finally, the time complexity of estimating $COR = DA$ is $3p^k(p + p + \cdots + p) = 3kp^{k+1}$. Compared with $p^k \times p^k$, this is a significant acceleration.

4 Generalization of DL Cryptanalysis to $\mathbb{F}_p$

DL cryptanalysis has been introduced by Langford and Hellman [32]. They find that if the cipher E can be decomposed as two parts $E_1 \circ E_0$ with high linear probability and differential probability respectively, then E_1 and E_0 can be combined into an efficient DL distinguisher for the whole cipher E. This attack depends on some assumptions, such as the independence between the two parts E_1 and E_0. In particular, the bias of the linear approximation in E_1 is not affected by the fact that they are applied to two intermediate values which correspond to plaintexts with a fixed difference. Then, a series of works [6,8,10,14,15,18,33,34] has been proposed to relieve the dependency and enhance the accuracy of the estimation of the distinguisher. Among which the DLCT is a novel tool introduced by Bar-On *et al.* [6], it shows that the cipher E can be decomposed into three parts $E_1 \circ E_m \circ E_0$, where DLCT of E_m part is used to cover the dependency for the connection. With all these dedicated researches, DL cryptanalysis has been developed into a more mature and powerful cryptanalytic method over $\mathbb{F}_2^n$. In the following, we first formalize the DL cryptanalysis over $\mathbb{F}_p$, then show that the construction of the DLCT over $\mathbb{F}_p$ also can be accelerated by using the FFT technique.

Let E be a cipher over $\mathbb{F}_p^t$ that can be decomposed into $E = E_1 \circ E_m \circ E_0$ and we consider these three parts separately as below.

- For E_0, it has a differential $\Delta_{in} \xrightarrow{p_0} \Delta_{out}$ with

$$DP_{E_0}[\Delta_{in} \rightarrow \Delta_{out}].$$

- For E_1, it has a linear approximation $\Gamma_{in} \xrightarrow{cor_{E_1}} \Gamma_{out}$ with

$$cor_{E_1} = EX(e^{\frac{2\pi\sqrt{-1}}{p}(\Gamma_{out} \cdot E_1(x) - \Gamma_{in} \cdot x)}).$$

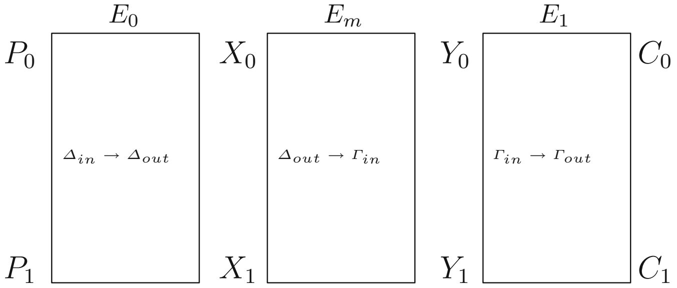

E_0
E_m
E_1
P_0
X_0
Y_0
C_0
$\Delta_{in} \rightarrow \Delta_{out}$
$\Delta_{out} \rightarrow \Gamma_{in}$
$\Gamma_{in} \rightarrow \Gamma_{out}$
P_1
X_1
Y_1
C_1

Fig. 1. Differential-linear distinguisher with a middle layer.

- For E_m, it has a difference-mask pair $\Delta_{out} \xrightarrow{cor_{E_m}} \Gamma_{in}$ with

$$cor_{E_m} = EX_{(x_0 - x_1 = \Delta_{out})}(e^{\frac{2\pi\sqrt{-1}}{p}(\Gamma_{in} \cdot E_m(x_0) - \Gamma_{in} \cdot E_m(x_1))}).$$

The relation of these three parts for a differential-linear distinguisher is depicted in Fig. 1. In order to distinguish the target cipher E from a random permutation, the adversary can prepare enough plaintext pairs (P_0, P_1) such that $P_0 - P_1 = \Delta_{in}$ and then check the correlation of $C_0 \cdot \Gamma_{out} - C_1 \cdot \Gamma_{out}$, where (C_0, C_1) is the corresponding ciphertext pairs of (P_0, P_1). Denote the intermediate values between E_0 and E_m by (X_0, X_1), E_m and E_1 by (Y_0, Y_1). Now, we consider how to evaluate the correlation of $C_0 \cdot \Gamma_{out} - C_1 \cdot \Gamma_{out}$ over $\mathbb{F}_p^t$ and give the following theorem.

Theorem 3. *Assume that the three parts of the cipher E are independent from each other, then the correlation of $C_0 \cdot \Gamma_{out} - C_1 \cdot \Gamma_{out}$ under a difference-mask pair $(\Delta_{out}, \Gamma_{in})$ of the cipher E over $\mathbb{F}_p^t$ is*

$$cor_{\Delta_{out}, \Gamma_{in}}(C_0 \cdot \Gamma_{out} - C_1 \cdot \Gamma_{out}) = -DP_{E_0}[\Delta_{in} \rightarrow \Delta_{out}] \times cor_{E_m} \times cor_{E_1}^2$$

Proof. Under the assumption that the two parts of the cipher E are independent from each other, we denote $cor_{\Delta_{out}, \Gamma_{in}}(C_0 \cdot \Gamma_{out} - C_1 \cdot \Gamma_{out})$ as cor' and $e^{\frac{2\pi\sqrt{-1}}{p}}$ as ω and have the following formula

$$\begin{aligned}
cor' &= EX(\omega^{(C_0 \cdot \Gamma_{out} - C_1 \cdot \Gamma_{out})}) \\
&= EX(\omega^{[(C_0 \cdot \Gamma_{out} - Y_0 \cdot \Gamma_{in}) - (C_1 \cdot \Gamma_{out} - Y_1 \cdot \Gamma_{in}) + (Y_0 \cdot \Gamma_{in} - Y_1 \cdot \Gamma_{in})]}) \\
&= EX(\omega^{(C_0 \cdot \Gamma_{out} - Y_0 \cdot \Gamma_{in})}) EX(\omega^{(Y_1 \cdot \Gamma_{in} - C_1 \cdot \Gamma_{out})}) EX(\omega^{(Y_0 \cdot \Gamma_{in} - Y_1 \cdot \Gamma_{in})}) \\
&= cor_{E_1} \times \overline{cor_{E_1}} \times EX(\omega^{(Y_0 \cdot \Gamma_{in} - Y_1 \cdot \Gamma_{in})}) \\
&= -DP_{E_0}[\Delta_{in} \rightarrow \Delta_{out}] \times EX(\omega^{(E_m(X_0) \cdot \Gamma_{in} - E_m(X_0 - \Delta_{out}) \cdot \Gamma_{in})}) \times |cor_{E_1}|^2 \\
&= -DP_{E_0}[\Delta_{in} \rightarrow \Delta_{out}] \times cor_{E_m} \times |cor_{E_1}|^2.
\end{aligned}$$

Thus, the correlation of the DL distinguisher can be derived from the corresponding three parts mentioned above.

Then the calculation of linear probability of a DL distinguisher can be obtained as below.

Corollary 1. *The linear probability of $C_0 \cdot \varGamma_{out} - C_1 \cdot \varGamma_{out}$ under a difference-mask pair $(\varDelta_{out}, \varGamma_{in})$ of the cipher E over $\mathbb{F}_p^t$ is*

$$LP_{\varDelta_{out},\varGamma_{in}}(C_0 \cdot \varGamma_{out} - C_1 \cdot \varGamma_{out}) = DP_{E_0}^2[\varDelta_{in} \to \varDelta_{out}] \times |cor_{E_m}|^2 \times |cor_{E_1}|^4.$$

Proof. By Definition 2 and Theorem 3, the proof can be ended.

According to [6], the correlation of cor_{E_m} is defined as a differential-linear connectivity table. We now adapt DLCT to the prime field as follows.

Definition 7 (DLCT over $\mathbb{F}_p$). *Let $E_S : \mathbb{F}_p^t \to \mathbb{F}_p^t$ be a function. The DLCT of E_S is a $p^t \times p^t$ table with rows corresponding to input differences of E_S and columns corresponding to output masks of E_S. For a given difference-mask pair $(\varDelta, \varGamma) \in \mathbb{F}_p^t \times \mathbb{F}_p^t$, its entry of DLCT is defined as*

$$DLCT_{E_S}(\varDelta, \varGamma) \triangleq \sum_{x_0 - x_1 = \varDelta} e^{\frac{2\pi\sqrt{-1}}{p}(\varGamma \cdot E_S(x_0) - \varGamma \cdot E_S(x_1))}.$$

Now for the E_m part, we can use the DLCT entry to represent its correlation by

$$p^t \times cor_{E_m} = DLCT_{E_m}.$$

When the E_m part is t parallel S-boxes, we can split the calculation of $DLCT_{E_m}$ into the calculation of $DLCT$ for each S-box. For the i-th S-box S_i, calculate $DLCT_{S_i}$. Assuming that at least one of $\varDelta[i]$ and $\varGamma[i]$ is zero, we have

$$DLCT_{S_i}(0, \varGamma[i]) = DLCT_{S_i}(\varDelta[i], 0) = p,$$

Then a property of DLCT over $\mathbb{F}_p^t$ is that for such special E_m over $\mathbb{F}_p^t$, given the complementary input difference $\varDelta \in \mathbb{F}_p^t$ and output mask $\varGamma \in \mathbb{F}_p^t$, where at most one of $\varDelta[i]$ and $\varGamma[i]$ is nonzero for each dimension i $(0 \le i < t)$, then it has

$$DLCT_{E_m}(\varDelta, \varGamma) = p^t.$$

Thus, it will pass with linear probability being 1 for E_m part.

4.1 Relation Between DLCT and DDT Under the FFT

Inspired by the work [6], DLCT also can be constructed efficiently using the Fourier Transform over $\mathbb{F}_2^n$, and we will discuss the relation between the DLCT and DDT over $\mathbb{F}_p^t$ in the following.

v*The Fourier Transform of Vector Functions.* Let $f : \mathbb{F}_p^t \to \mathbb{C}$ be a complex valued function. The Fourier Transform of f is the function $\hat{f} : \mathbb{F}_p^t \to \mathbb{C}$ defined by

$$\hat{f}(x) = \frac{1}{p^t} \sum_{y \in \mathbb{F}_p^t} f(y) \times e^{\frac{2\pi\sqrt{-1}}{p} x \cdot y}.$$

The DDT. For a vectorial function $S : \mathbb{F}_p^t \to \mathbb{F}_p^t$, the DDT of S is a $p^t \times p^t$ table whose rows correspond to input differences of S and whose columns correspond to output differences of S. Formally, for $\Delta_{in} \in \mathbb{F}_p^t$ and $\Delta_{out} \in \mathbb{F}_p^t$, we have

$$DDT_S(\Delta_{in}, \Delta_{out}) = \#\{x | S(x) - S(x - \Delta_{in}) = \Delta_{out}\}.$$

It can be observed that each row of the DLCT is equal to the Fourier Transform of the corresponding row of the DDT. Formally, for each $\Delta \in \mathbb{F}_p$, denote the function corresponding to the row with index Δ of the DDT by f_Δ. That is, $f_\Delta : \mathbb{F}_p^t \to \mathbb{R}$ is defined by

$$f_\Delta(\Delta') = DDT_S(\Delta, \Delta').$$

Proposition 2. *For any $\Gamma \in \mathbb{F}_p^t$, we have $DLCT_S(\Delta, \Gamma) = p^t \times \hat{f_\Delta}(\Gamma)$.*

Proof. By the definitions of the DLCT and of the FWT, we have

$$\begin{aligned} DLCT_S(\Delta, \Gamma) = \sum_{x_0 - x_1 = \Delta} e^{\frac{2\pi\sqrt{-1}}{p}(\Gamma \cdot S(x_0) - \Gamma \cdot S(x_1))} &= \sum_{x_0 - x_1 = \Delta} e^{\frac{2\pi\sqrt{-1}}{p}\Gamma \cdot (S(x_0) - S(x_1))} \\ &= \sum_{y \in \mathbb{F}_p^t} e^{\frac{2\pi\sqrt{-1}}{p}\Gamma \cdot y} \times DDT_S(\Delta, y) = p^t \times \hat{f_\Delta}(\Gamma), \end{aligned}$$

as claimed.

Therefore, in order to generate a row of DLCT, we first need to generate the corresponding row of DDT with the time complexity of $\mathcal{O}(p^t)$. Proposition 2 implies that the next step is to perform the FFT on each row of DDT to obtain the corresponding row of DLCT,

$$\begin{aligned} \begin{pmatrix} DLCT_S(\Delta, 0^{\mathcal{V}}) \\ DLCT_S(\Delta, 1^{\mathcal{V}}) \\ \vdots \\ DLCT_S(\Delta, (p^t - 1)^{\mathcal{V}}) \end{pmatrix} &= p^t \begin{pmatrix} \hat{f_\Delta}(0^{\mathcal{V}}) \\ \hat{f_\Delta}(1^{\mathcal{V}}) \\ \vdots \\ \hat{f_\Delta}((p^t - 1)^{\mathcal{V}}) \end{pmatrix} \\ &= p^t \begin{pmatrix} \omega^{0^{\mathcal{V}} \cdot 0^{\mathcal{V}}} & \omega^{0^{\mathcal{V}} \cdot 1^{\mathcal{V}}} & \cdots & \omega^{0^{\mathcal{V}} \cdot (p^t-1)^{\mathcal{V}}} \\ \omega^{1^{\mathcal{V}} \cdot 0^{\mathcal{V}}} & \omega^{1^{\mathcal{V}} \cdot 1^{\mathcal{V}}} & \cdots & \omega^{1^{\mathcal{V}} \cdot (p^t-1)^{\mathcal{V}}} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(p^t-1)^{\mathcal{V}} \cdot 0^{\mathcal{V}}} & \omega^{(p^t-1)^{\mathcal{V}} \cdot 1^{\mathcal{V}}} & \cdots & \omega^{(p^t-1)^{\mathcal{V}} \cdot (p^t-1)^{\mathcal{V}}} \end{pmatrix} \begin{pmatrix} f_\Delta(0^{\mathcal{V}}) \\ f_\Delta(1^{\mathcal{V}}) \\ \vdots \\ f_\Delta((p^t - 1)^{\mathcal{V}}) \end{pmatrix} \\ &= p^{\frac{3t}{2}} F \begin{pmatrix} f_\Delta(0^{\mathcal{V}}) \\ f_\Delta(1^{\mathcal{V}}) \\ \vdots \\ f_\Delta((p^t - 1)^{\mathcal{V}}) \end{pmatrix}, \end{aligned}$$

where $\omega = e^{\frac{2\pi\sqrt{-1}}{p}}$ and F is the DFT matrix in Theorem 2. For any $i \in \{0, 1, \cdots, p^t - 1\}$, $i^{\mathcal{V}}$ means that express i as a vector of length t in p-based number. According to the property of F, the complexity of this step is $\mathcal{O}(tp^{t+1})$. So the time complexity of generating the entire DLCT is $\mathcal{O}(tp^{2t+1})$. This significantly improves over the trivial generating method which requires $\mathcal{O}(p^{3t})$ operations.

5 Applications to HADESMiMC

In this section, as applications of our generalizations of these cryptanalytic methods presented above, we concentrate on a target cipher—HADESMiMC-128.

5.1 Description of HADES

As is demonstrated in Fig. 6 (see Appendix A), a block cipher designed according to the HADES strategy has the following three operations:

- AddRoundKey—$ARK(\cdot)$, it represents the addition of a round subkey to the state.
- S-box layer—$S(\cdot)$, it represents the application of the S-box S with the state. Note that the S-box will only be applied to a part of the state in the middle rounds, which is denoted by $S^*(\cdot)$.
- Linear layer—$MDS(\cdot)$, it represents the application of the linear mixing matrix with the state.

Under this construction, a keyed permutation HADESMiMC has been proposed, it has $R = 2R_f + R_P$ rounds in total, in which $2R_f$ denotes the outer rounds and R_P denotes the inner rounds. For a full round $R_k(\cdot) : \mathbb{F}_p^t \rightarrow \mathbb{F}_p^t$ is defined as

$$R_k(\cdot) = k + M \times S(\cdot),$$

where t is the branch number, $k \in \mathbb{F}_p^t$ is the round subkey, $M \in \mathbb{F}_p^{t\times t}$ is an MDS matrix generated by a Cauchy matrix, and $S(\cdot) : \mathbb{F}_p^t \rightarrow \mathbb{F}_p^t$ is the non-linear layer. Similarly, for a partial round $R_k^*(\cdot) : \mathbb{F}_p^t \rightarrow \mathbb{F}_p^t$ is defined as

$$R_k^*(\cdot) = k + M \times S^*(\cdot).$$

Note that the last round does not contain the MDS matrix.

In this paper, we mainly focus on the version with the SHARK-like security. It has $t = 16$, and each partial round has 1 S-box and 15 identity functions. As for the S-box $S(x) \equiv x^3(mod\ p)$, according to Hermite's criterion, S is a permutation over $\mathbb{F}_p$ iff $gcd(3, p-1) = 1$, thus the chosen prime $p = 251$. In [30], the authors find that not every Cauchy matrix provides the same security level when considering algebraic attacks. But, as we only focus on statistical attacks, we randomly select a Cauchy matrix when performing the security analysis in this paper, which is given in Appendix A. In order to avoid confusion of names, we temporarily denote it as HADESMiMC-128 in this paper.

Key Schedule. Let $k = [k_0, \cdots, k_{t-2}, k_{t-1}] \in \mathbb{F}_p^t$ be the master key, the whitening key $AK_0 \in \mathbb{F}_p^t$ in the first round is equal to k. HADESMiMC-128 adopts a linear key schedule as below

$$AK_i = \hat{M} \times AK_{i-1} + RC_i, \qquad i = 1, \cdots, R.$$

AK_i is the subkey of i-th round and RC_i is the round constant, $\hat{M}$ is a t by t MDS matrix with $M = \hat{M}$ allowed.

5.2 Search Models for Distinguishers over $\mathbb{F}_p$

Automatic search has been widely used in the cryptanalysis of the symmetric-key primitives over $\mathbb{F}_2^n$ [31,36–38]. To construct the search models for differential and linear mask over the prime field $\mathbb{F}_p$, we should divide the target cipher into several basic operations firstly, then describe the exact propagation of the differential and mask behaviors for each operation. In this paper, we utilize the SAT/SMT [7,20] techniques for searching the distinguisher. The detailed SAT/SMT based model for determining a lower bound of the linear probability is described as follows. Note that each variable in the following constraints is expressed as a bit vector. To be more precise, for a variable $\Gamma \in \mathbb{F}_p$, Γ can be expressed by using $\lceil \log_2 p \rceil$ bits. We provide the codes in https://www.dropbox.com/s/w1cylrp7r6s0j1y/search_model_linear.zip?dl=0.

Constraints Imposed by Modular Addition. The modular addition maps $(x, y) \in \mathbb{F}_p^2$ to $z = x + y \ mod \ p$. Let Γ_{in}^1 and Γ_{in}^2 represent two input masks for modular addition, the output mask is Γ_{out}. Then the approximation of modular addition due to $(\Gamma_{in}^1, \Gamma_{in}^2, \Gamma_{out})$ is of nonzero-correlation if and only if it fulfills $\Gamma_{in}^1 = \Gamma_{in}^2 = \Gamma_{out}$.

Constraints Imposed by Linear Transformation. A linear transformation with matrix representation M maps x to $y = Mx$. Let Γ_{in} and Γ_{out} represent the input and output masks of x and y. Then the approximation of M due to $(\Gamma_{in}, \Gamma_{out})$ is of nonzero-correlation if and only if it fulfills $\Gamma_{in} = M^{\mathrm{T}}\Gamma_{out}$.

Constraints Imposed by k-Branch ($k \geq 3$). The k-branch operation maps $x \in \mathbb{F}_p$ to $(x_1, \cdots, x_{k-1}) \in \mathbb{F}_p^{k-1}$ with $x = x_1 = \cdots = x_{k-1}$. Let Γ_{in} represent the input mask for k-branch, the output masks are Γ_{out}^1, Γ_{out}^2, $\cdots$, Γ_{out}^{k-1}. Then the approximation of k-branch due to $(\Gamma_{in}, \Gamma_{out}^1, \cdots, \Gamma_{out}^{k-1})$ is of nonzero-correlation if and only if it fulfills $\Gamma_{in} = \Gamma_{out}^1 + \Gamma_{out}^2 + \cdots + \Gamma_{out}^{k-1}$.

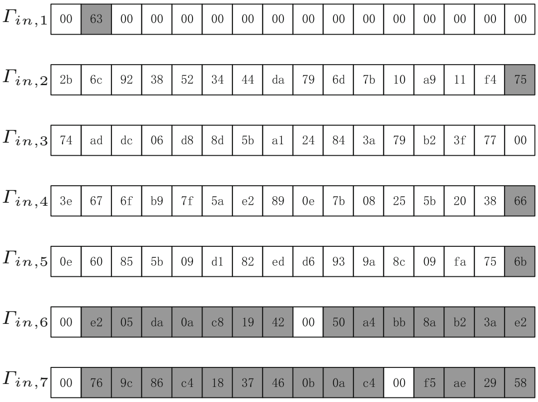
Constraints Describing the S-Box Operation. Suppose Γ_{in} and Γ_{out} are the input and output masks of a bijective S-box S, which is defined over $\mathbb{F}_p$. All these informations are derived from the linear approximation table (LAT) of S, which is a $p \times p$ table and $\forall(\Gamma_{in}, \Gamma_{out}) \in \mathbb{F}_p \times \mathbb{F}_p$, the LAT entry for the pair $(\Gamma_{in}, \Gamma_{out})$ is $LAT(\Gamma_{in}, \Gamma_{out}) = LP_S(\Gamma_{in}, \ \Gamma_{out})$, In practice, the S-box, especially x^3, has a large range. When $p = 251$, except for the first row and the first column, the values in each row range from 2^{-6} to 2^{-26}. Hence, we use $B[\Gamma_{in}, \Gamma_{out}]$ as an indicator to represent the linear probability and whether it is a possible differential transition of S,

$$\begin{cases} B[\Gamma_{in}, \Gamma_{out}] = \lceil -\log_2(LAT(\Gamma_{in}, \Gamma_{out})) \rceil, & LAT(\Gamma_{in}, \Gamma_{out}) \neq 0 \\ B[\Gamma_{in}, \Gamma_{out}] = -1, & LAT(\Gamma_{in}, \Gamma_{out}) = 0 \end{cases}$$

Objective Function. Now, under the condition that $B \neq -1$, we set up the objective function to be the sum of all indicators of all S-boxes. It corresponds to a rough estimate of the linear probability, and constraints can be added to determine the lower bound of the distinguisher. When searching for linear distinguishers, we round the probability of S-box. Therefore, in order to obtain an accurate linear probability, we need to calculate it again with the actual LAT.

The construction of searching differential model is similar to that of linear, and the detailed description is in Appendix C.

5.3 Linear and Differential-Linear Distinguishers of HADESMiMC-128



$\Gamma_{in,1}$	00	**63**	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$\Gamma_{in,2}$	2b	6c	92	38	52	34	44	da	79	6d	7b	10	a9	11	f4	**75**
$\Gamma_{in,3}$	74	ad	dc	06	d8	8d	5b	a1	24	84	3a	79	b2	3f	77	00
$\Gamma_{in,4}$	3e	67	6f	b9	7f	5a	e2	89	0e	7b	08	25	5b	20	38	**66**
$\Gamma_{in,5}$	0e	60	85	5b	09	d1	82	ed	d6	93	9a	8c	09	fa	75	**6b**
$\Gamma_{in,6}$	00	**e2**	**05**	**da**	**0a**	**c8**	**19**	**42**	00	**50**	**a4**	**bb**	**8a**	**b2**	**3a**	**e2**
$\Gamma_{in,7}$	00	**76**	**9c**	**86**	**c4**	**18**	**37**	**46**	**0b**	**0a**	**c4**	00	**f5**	**ae**	**29**	**58**

Fig. 2. 6-Round linear distinguisher.

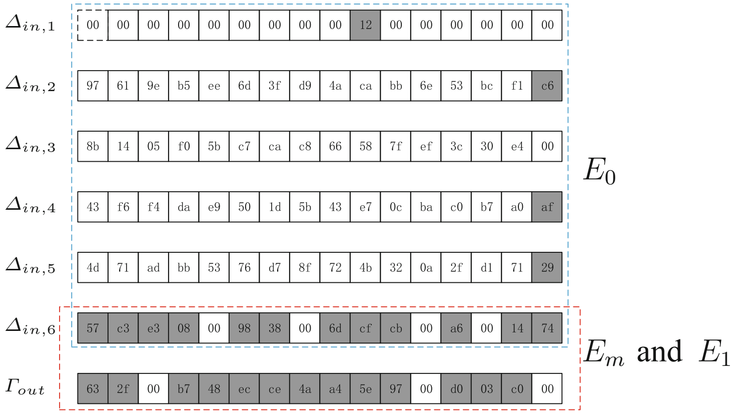
6-Round Linear Distinguisher. The best linear approximation we found reaches 6 rounds, as shown in Fig. 2. According to the linear approximation table (LAT) of $S(x) = x^3$ over $\mathbb{F}_{251}$, the linear probability of this approximation is evaluated as $2^{-119.9}$.

6-Round DL Distinguisher. As shown in Fig. 3, we give a better DL distinguisher. The linear probability can be calculated according to Corollary 1, which is $2^{-55.77}$.

Considering the non uniqueness of prime and MDS matrix, we also tested other primes, such as $p \in \{239, 233, 227\}$, and randomly select multiple MDS matrices under each p, the results show that we can find linear trails with similar patterns. The experimental results are presented in Appendix D.

5.4 Linear Attacks on Round Reduced HADESMiMC-128

We now mount key recovery attacks and show that our generalizations of the FFT technique for correlation estimations over $\mathbb{F}_p$ can contribute to reduce the complexity and lead to better attacks. Before the attack, we set $\alpha = 8$, then $e^{-\frac{\alpha}{2}} = 2^{-5.77}$. In this way, the success rate of key recovery attack is $P_s = 1 - e^{-\frac{\alpha}{2}} = 98\%$, and the amount of data is $N = 2^4 \times LP^{-1}$.



$\Delta_{in,1}$	00	00	00	00	00	00	00	00	00	12	00	00	00	00	00	00
$\Delta_{in,2}$	97	61	9e	b5	ee	6d	3f	d9	4a	ca	bb	6e	53	bc	f1	c6
$\Delta_{in,3}$	8b	14	05	f0	5b	c7	ca	c8	66	58	7f	ef	3c	30	e4	00
$\Delta_{in,4}$	43	f6	f4	da	e9	50	1d	5b	43	e7	0c	ba	c0	b7	a0	af
$\Delta_{in,5}$	4d	71	ad	bb	53	76	d7	8f	72	4b	32	0a	2f	d1	71	29
$\Delta_{in,6}$	57	c3	e3	08	00	98	38	00	6d	cf	cb	00	a6	00	14	74
Γ_{out}	63	2f	00	b7	48	ec	ce	4a	a4	5e	97	00	d0	03	c0	00

Fig. 3. 6-Round differential-linear distinguisher.

6-Round Key Recovery Attack. We cut out the last 5 rounds in Fig. 2 as a new distinguisher. That is, $LP(\Gamma_{out,1}, \Gamma_{out,6}) = 2^{-113.9}$. We add one full S-box round in the head to complete the 6-round key recovery attack. The output mask of the first round has one active word, so we can recover the information on one whitening key word.

According to the general framework in Sect. 3.1, the time complexity of recover one word of the whitening key of the first round is $p \times p = 2^{15.94}$ one-round encryptions. According to the α we select, the number of keys left in this step is $p \times e^{-\frac{\alpha}{2}} \approx 5$. Then these keys and the remaining 15 words of the whitening key can be obtained by exhaustive search with about $5 \times p^{15} = 2^{121.89}$ 6-round encryptions. In all, 6-round of HADES-128 can be attacked with $2^{117.9}$ known plaintexts and $2^{117.9} + 2^{121.89} = 2^{121.98}$ 6-round encryptions.

7-Round Key Recovery Attack with the FFT Technique. As shown in Fig. 4, we adopt the 6-round linear trail as the distinguisher with $LP(\Gamma_{in,1}, \Gamma_{in,7}) = 2^{-119.9}$. One full S-box round in the tail is added to lanuch the 7-round key recovery attack. The input mask of the 7-th round has 14 active words, so we can recover 14 words of the subkey AK_7 of the last round. It should be noted that if we do not use the FFT technique, the time complexity required to recover these 14 subkey words is $p^{14} \times p^{14} = 2^{223.2} > p^t = 2^{127.54}$.

According to the fast algorithm given in Sect. 3.2, the time complexity of recover 14 words of AK_7 is $3 \times 14 \times p^{15} = 2^{124.9}$ one-round encryptions, which is equivalent to $2^{124.9} \times \frac{14}{3 \times 16 + 4} = 2^{123}$ 7-round encryptions. According to the α we select, the number of keys left in this step is $p^{14} \times e^{-\frac{\alpha}{2}} = 2^{105.83}$. Then these subkeys and the remaining 2-word subkey can be obtained by exhaustive search with about $2^{105.83} \times p^2 = 2^{121.77}$ 7-round encryptions. Thus, 7-round of HADESMiMC-128 can be attacked with $2^{123.9}$ known plaintexts and $2^{123.9} + 2^{123} + 2^{121.77} = 2^{124.72}$ 7-round encryptions.

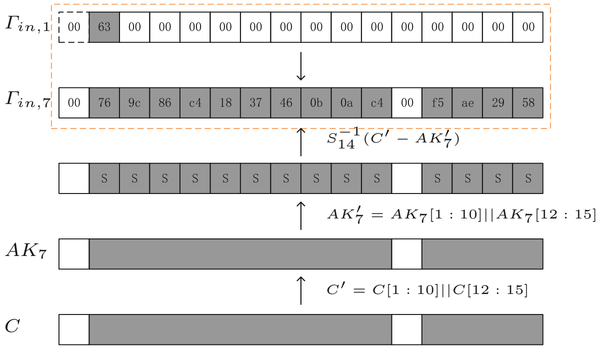

$\Gamma_{in,1}$
00 63 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$\Gamma_{in,7}$
00 76 9c 86 c4 18 37 46 0b 0a c4 00 f5 ae 29 58
$S_{14}^{-1}(C' - AK'_7)$
S S S S S S S S S S S S S S
$AK'_7 = AK_7[1:10]||AK_7[12:15]$
AK_7
$C' = C[1:10]||C[12:15]$
C

Fig. 4. 7-Round linear attack.

5.5 7-Round Differential-Linear Attack on HADESMiMC-128

Similar to linear attacks, the time complexity of DL attack can be roughly estimated as the ratio of the guessed key amount to the linear probability of the distinguisher. As shown in Fig. 5, we add one full S-box round (the seventh round) after E_m and E_1 to form a 7-round trail, where the used DL distinguisher is with linear probability $LP = 2^{-55.77}$. There are 13 active S-boxes in the last round, if we guess all 13 words of the subkey AK_7, this leads an invalid attack with time complexity $2^{55.77} \times 251^{13} = 2^{159.4}$.

In order to reduce the complexity, we need to guess fewer subkey words. In fact, the number of guessed subkey words is required to be less or equal than 8, due to $2^{55.77} \times p^8 = 2^{119.54} < 2^{127.54}$. Assume that the output mask is active at position i, then for a pair of word $(C_0[i], C_1[i])$ of a ciphertext pair (C_0, C_1), we have to guess the subkey word $AK_7[i]$ to complete operation $S^{-1}(C_0[i] - AK_7[i]) - S^{-1}(C_1[i] - AK_7[i])$. However, if the condition $C_0[i] = C_1[i]$ can be guranteed when generating ciphertext pairs, then it will return 0 directly without guessing $AK_7[i]$ to decrypt, but this will also consume more data to satisfy the conditions on the ciphertext pair. Fortunately, by constructing more plaintext pairs, we can filter out enough ciphertext pairs that meet the conditions. For example, suppose that the attack needs N plaintext pairs satisfying the input difference. In order to guess one key word less, we need to construct pN plaintext pairs that satisfy the input difference to filter out N ciphertext pairs that meet the conditions. The details of the key recovery attack are listed as below.

- Set $\alpha = 32$, then $e^{-\frac{\alpha}{2}} = 2^{-23.08}$, $P_s = 99.99\%$ and we need $N = 2\alpha \times LP^{-1} = 2^{61.77}$ ciphertext pairs.
- We first guess 7 subkey words, which requires $p^{13-7} \times 2^{61.77} = 2^{109.63}$ plaintext pairs to filter out $2^{61.77}$ such kind of ciphertext pairs. During the filtering, we need to perform $2^{109.63}$ comparison operations, and each operation can be equivalent to one S-box operation. There are $3 \times 16 + 4 = 52$ S-boxes in the 7-round algorithm, so the time complexity of this step is $2^{109.63} \times \frac{1}{52} = 2^{103.93}$ 7-round encryptions.

- Then, we can recover 6 subkey words from these pairs. For this step, the time complexity is $2 \times 2^{61.77} \times p^7 = 2^{118.6}$ parity computations. According to the α we select, the number of keys left in this step is $p^7 \times e^{-\frac{\alpha}{2}} = 2^{32.72}$. Then these subkeys and the remaining 9 words of the subkey can be obtained by exhaustive search with about $2^{32.72} \times p^9 = 2^{104.46}$ 7-round encryptions.

Finally, by using DL cryptanalysis, 7-round of HADESMiMC-128 can be attacked with $2^{110.63}$ chosen plaintexts and $2^{110.63}+2^{103.93}+2^{118.6}\times\frac{6}{52}+2^{104.46} = 2^{115.53}$ 7-round encryptions.

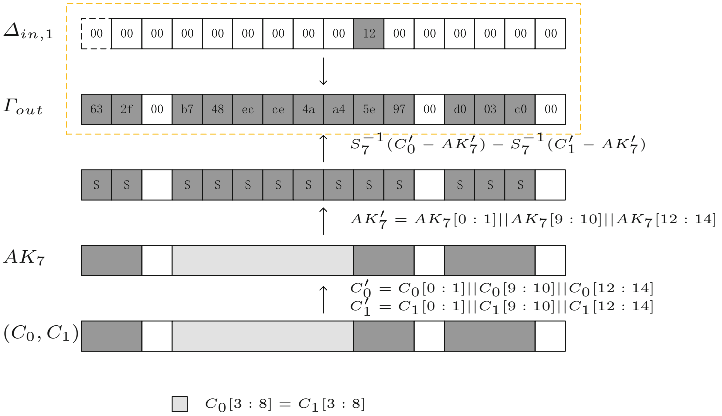

$\Delta_{in,1}$
00 00 00 00 00 00 00 00 00 12 00 00 00 00 00 00
Γ_{out}
63 2f 00 b7 48 ec ce 4a a4 5e 97 00 d0 03 c0 00
$S_7^{-1}(C_0' - AK_7') - S_7^{-1}(C_1' - AK_7')$
S S S S S S S S S S S S S
$AK_7' = AK_7[0:1]||AK_7[9:10]||AK_7[12:14]$
AK_7
$C_0' = C_0[0:1]||C_0[9:10]||C_0[12:14]$
$C_1' = C_1[0:1]||C_1[9:10]||C_1[12:14]$
(C_0, C_1)
$C_0[3:8] = C_1[3:8]$

Fig. 5. 7-Round differential-linear attack.

6 Conclusions

In this paper, we showed that the FFT technique for the estimations of correlation can be applied to the fields with odd characteristic and significantly improves the complexity of Matsui's *Algorithm 2* over $\mathbb{F}_p$. Then, we generalized the DL cryptanalysis to $\mathbb{F}_p$. We also included the DLCT into the generalization, and found the relation between DLCT and DDT over $\mathbb{F}_p$. Finally, as applications of the FFT-based linear attacks and DL attacks over $\mathbb{F}_p$, we mounted key recovery attacks on 7-round HADESMiMC-128.

Further Discussion. The main work of this paper is to obtain the theoretical framework. We show the effectiveness of the theory under 8-bit prime number, and we also note that the search of the distinguisher under large prime number is difficult. Therefore, if the distinguisher search technique under large prime numbers can be further improved, then more ciphers can be evaluated. This is an interesting question for future research.

Acknowledgement. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This research is supported by the National Key Research and Development Program of China (Grant No. 2018YFA0704702), the National Natural Science Foundation of China (Grant No. 62032014), the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025). Puwen Wei is partially supported by National Key R&D Program of China (Grant No. 2022YFB2701700), Shandong Provincial Natural Science Foundation (Grant No. ZR2020MF053). Shiyao Chen is supported by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

A Constructions of HADES and MDS

MDS Matrices. All MDS matrices in the HADES strategy are Cauchy matrices.

Definition 8 (Cauchy Matrix). *Let* $X = (x_1, \cdots, x_t) \in \mathbb{F}_p^t$ *and* $Y = (y_1, \cdots, y_t) \in \mathbb{F}_p^t$ *s.t.*

- $\forall i \neq j : x_i \neq x_j, y_i \neq y_j,$
- $\forall i, j \in \{1, 2, \cdots t\} : x_i + y_j \neq 0, x_i \neq y_j.$

Let M *be the Cauchy matrix corresponding to* (X, Y)*, then its entry at* (i, j) *is*

$$M_{i,j} = \frac{1}{x_i + y_j}.$$

A Cauchy matrix is an MDS matrix.

Practical Example. In the cryptanalysis of a concrete instance of HADESMiMC-128 working on $GF(251)$, we select a pair (X, Y) randomly as below,

$$X = [250, 171, 161, 235, 93, 225, 229, 123, 122, 106, 246, 43, 55, 90, 186, 39],$$

$$Y = [87, 9, 179, 81, 139, 35, 169, 61, 195, 217, 110, 125, 230, 76, 175, 248],$$

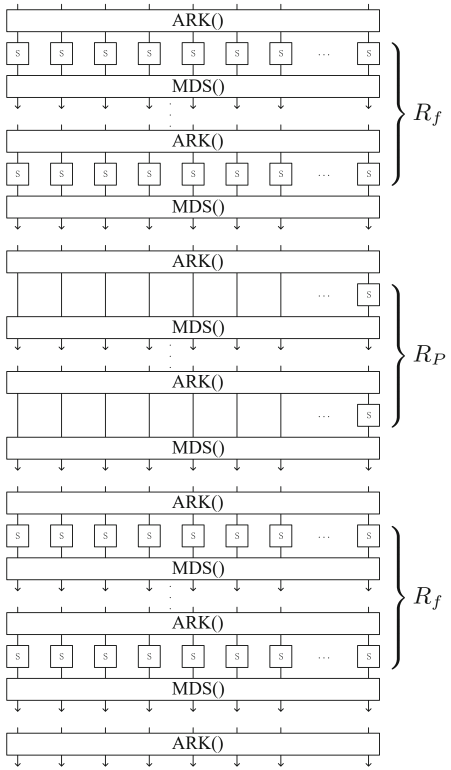


Fig. 6. Construction of HADES.

and get the following 16 × 16 Cauchy matrix,

$$\begin{pmatrix}
108 & 157 & 55 & 91 & 231 & 96 & 127 & 205 & 22 & 43 & 76 & 83 & 57 & 164 & 88 & 188 \\
36 & 152 & 71 & 1 & 234 & 145 & 110 & 66 & 227 & 11 & 159 & 106 & 82 & 188 & 37 & 127 \\
167 & 220 & 110 & 223 & 41 & 73 & 197 & 225 & 153 & 168 & 113 & 208 & 52 & 233 & 189 & 224 \\
99 & 215 & 77 & 112 & 100 & 185 & 105 & 106 & 122 & 5 & 243 & 76 & 156 & 205 & 30 & 66 \\
152 & 32 & 12 & 88 & 66 & 151 & 137 & 207 & 95 & 234 & 183 & 38 & 129 & 101 & 192 & 53 \\
107 & 59 & 105 & 178 & 20 & 28 & 165 & 208 & 101 & 46 & 3 & 71 & 16 & 246 & 219 & 225 \\
112 & 193 & 8 & 234 & 118 & 58 & 181 & 103 & 74 & 121 & 174 & 39 & 35 & 172 & 105 & 10 \\
202 & 116 & 64 & 16 & 137 & 224 & 49 & 236 & 15 & 110 & 237 & 167 & 32 & 111 & 235 & 228 \\
245 & 23 & 246 & 183 & 226 & 8 & 182 & 203 & 232 & 174 & 66 & 188 & 169 & 161 & 191 & 135 \\
238 & 227 & 96 & 200 & 209 & 162 & 136 & 248 & 246 & 129 & 43 & 138 & 189 & 40 & 159 & 39 \\
150 & 63 & 88 & 109 & 133 & 159 & 75 & 130 & 144 & 148 & 153 & 228 & 222 & 99 & 220 & 94 \\
56 & 140 & 225 & 83 & 40 & 177 & 148 & 70 & 193 & 28 & 105 & 127 & 194 & 135 & 38 & 182 \\
175 & 51 & 59 & 24 & 22 & 53 & 158 & 132 & 250 & 12 & 143 & 152 & 96 & 23 & 239 & 140 \\
78 & 71 & 14 & 160 & 57 & 249 & 157 & 128 & 96 & 130 & 187 & 244 & 211 & 62 & 18 & 176 \\
194 & 121 & 240 & 204 & 173 & 92 & 70 & 188 & 56 & 180 & 106 & 205 & 143 & 137 & 89 & 203 \\
2 & 68 & 38 & 228 & 55 & 173 & 35 & 123 & 59 & 201 & 219 & 75 & 14 & 227 & 156 & 7
\end{pmatrix}$$

B Kronecker Product

Definition 9 (Kronecker Product $\otimes$). *Assume A is a matrix of size $m \times n$ and B is a matrix of size $r \times s$, then the Kronecker Product of A and B is a matrix C of size $mr \times ns$. In terms of formula :*

$$C = A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}$$

C The Construction of Searching Differential Model

Constraints Imposed by Modular Addition. Let Δ_{in}^1 and Δ_{in}^2 represent two input differences for modular addition, the output difference is Δ_{out}. Then the character of modular addition due to $(\Delta_{in}^1, \Delta_{in}^2, \Delta_{out})$ is of nonzero probability if and only if it fulfills $\Delta_{in}^1 + \Delta_{in}^2 = \Delta_{out}$.

Constraints Imposed by Linear Transformation. Let column vector Δ_{in} and Δ_{out} represent the input and output differences for linear transformation M. Then the character of M due to $(\Delta_{in}, \Delta_{out})$ is of nonzero probability if and only if it fulfills $\Delta_{out} = M \cdot \Delta_{in}$.

Constraints Imposed by k-Branch ($k \geq 3$). Let Δ_{in} represent the input difference for k-branch, the output differences are Δ_{out}^1, Δ_{out}^2, $\cdots$, Δ_{out}^{k-1}. The relation between these differences is $\Delta_{in} = \Delta_{out}^1 = \Delta_{out}^2 = \cdots = \Delta_{out}^{k-1}$.

Then the character of k-branch due to $(\Delta_{in}, \Delta_{out}^1, \cdots, \Delta_{out}^{k-1})$ is of nonzero probability if and only if it fulfills $\Delta_{in} = \Delta_{out}^1 = \Delta_{out}^2 = \cdots = \Delta_{out}^{k-1}$.

Constraints Describing the S-box Operation. Suppose Δ_{in} and Δ_{out} are the input and output differences of a bijective S-box S, which is defined over $\mathbb{F}_p$. Use $B[\Delta_{in}, \Delta_{out}]$ as an indicator to represent whether it is a possible differential transitions of S. When $B[\Delta_{in}, \Delta_{out}] = 1$, it means that $\Delta_{in} \xrightarrow{S} \Delta_{out}$ is a possible transitions. Otherwise, it means an impossible transitions. Then, we have the relations

$$\begin{cases} B[\Delta_{in}, \Delta_{out}] = 1, & DDT(\Delta_{in}, \Delta_{out}) \notin \{0, p\} \\ B[\Delta_{in}, \Delta_{out}] = 0, & DDT(\Delta_{in}, \Delta_{out}) = p \\ B[\Delta_{in}, \Delta_{out}] = -1, & DDT(\Delta_{in}, \Delta_{out}) = 0 \end{cases}$$

Objective Function. Now, under the condition that $B \neq -1$, we set up the objective function to be the sum of all indicators of all S-boxes. It corresponds to the number of active S-boxes, and can be added constraints to determine a lower bound of the distinguisher. Note that in the block cipher over $\mathbb{F}_p$, the S-box is generally designed by using the power map, that is x^3 and $x \in \mathbb{F}_p$. Except the first row and first column for $(\Delta_{in}, \Delta_{out}) = (0, 0)$, the DDT of x^3 is kind of balanced, that is, the half of the entries are 2 and just one entry is 1 in each row and column. Thus, for any active S-box of x^3, it almost has the probability $\frac{2}{p}$. Naturally, we can directly count the number of S-boxes of the distinguisher to simplify the probability representation in the model for this kind of S-box.

D Linear Trails Under Other Primes and MDS Matrices

For each $p \in \{227, 233, 239\}$, we randomly select 4 MDS matrices.
$p = 227$.

- Matrix1:

$$X = [205, 212, 217, 137, 101, 65, 133, 199, 126, 83, 178, 158, 107, 37, 55, 216],$$

$$Y = [52, 201, 25, 146, 159, 220, 114, 66, 182, 98, 135, 47, 197, 87, 54, 169].$$

- Matrix2:

$$X = [163, 154, 18, 168, 141, 203, 217, 132, 221, 206, 55, 19, 130, 209, 72, 7],$$

$$Y = [215, 178, 148, 156, 57, 32, 15, 58, 138, 152, 118, 133, 224, 116, 60, 68].$$

- Matrix3:

$$X = [152, 161, 106, 200, 172, 60, 94, 179, 20, 160, 176, 164, 195, 50, 187, 193],$$

$$Y = [198, 91, 115, 6, 183, 217, 41, 221, 219, 22, 101, 28, 148, 9, 88, 175].$$

- Matrix4:

$$X = [226, 148, 161, 24, 91, 116, 3, 16, 222, 107, 14, 65, 102, 106, 200, 20],$$

$$Y = [174, 73, 163, 95, 58, 188, 146, 176, 205, 9, 132, 217, 155, 189, 122, 11].$$

(See Tables 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 and 13).

Table 2. 6-round linear distinguisher ($p = 227$, matrix1) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	5c, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00	a4, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
2	c9, 8e, 01, 8c, 72, 33, 3a, 71, 42, b5, b2, 78, aa, 4f, 3b, 66	c9, 8e, 01, 8c, 72, 33, 3a, 71, 42, b5, b2, 78, aa, 4f, 3b, 78
3	2a, 8a, 54, 97, 18, 9c, d3, 11, b3, 0d, 69, d5, 98, 20, 47, 67	2a, 8a, 54, 97, 18, 9c, d3, 11, b3, 0d, 69, d5, 98, 20, 47, 9e
4	df, 73, 75, ab, 34, 42, a5, e2, 7a, 3b, 36, 4e, 87, 48, b7, 67	df, 73, 75, ab, 34, 42, a5, e2, 7a, 3b, 36, 4e, 87, 48, b7, 5e
5	77, 94, c5, c6, 4a, c6, c2, 68, 04, 5a, 40, be, 31, 07, 79, bc	77, 94, c5, c6, 4a, c6, c2, 68, 04, 5a, 40, be, 31, 07, 79, e0
6	57, 19, 7b, b5, 2b, 73, bf, 40, b1, 33, ae, 87, b2, 00, 74, 77	3d, 50, 5a, 63, cf, dd, 92, 66, c4, aa, be, 3f, b1, 00, 70, 85
7	79, 3f, 81, d3, 00, 1b, 41, bc, 27, 5f, 7d, 00, a6, 1d, 16, 94	

$p = 233$.

Table 3. 6-round linear distinguisher ($p = 227$, matrix2) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 00, 00, 00, 00, 00, 5c, 00, 00, 00, 00, 00, 00, 00	00, 00, 00, 00, 00, 00, 00, 00, 0a, 00, 00, 00, 00, 00, 00, 00
2	ae, d2, 11, 14, 0f, 0a, b5, 70, 19, 1b, 81, 9a, aa, a2, 14, 23	ae, d2, 11, 14, 0f, 0a, b5, 70, 19, 1b, 81, 9a, aa, a2, 14, 0f
3	65, 4a, c7, b6, 58, 20, 00, b3, 7c, a2, 5a, 04, 43, 69, 8f, 96	65, 4a, c7, b6, 58, 20, 00, b3, 7c, a2, 5a, 04, 43, 69, 8f, 78
4	28, 69, 9b, c1, 9b, 46, 64, c0, 25, c9, 4d, af, 0a, ca, bb, 71	28, 69, 9b, c1, 9b, 46, 64, c0, 25, c9, 4d, af, 0a, ca, bb, 91
5	aa, 28, 8a, b6, 73, 9f, a4, 47, 0f, 1d, 37, 83, 64, 7e, de, cc	aa, 28, 8a, b6, 73, 9f, a4, 47, 0f, 1d, 37, 83, 64, 7e, de, d3
6	b3, 34, 79, c1, 00, c3, af, e0, b2, 81, 96, 29, 15, 95, 94, 75	d2, 1b, d3, 8f, 00, 2c, 44, 30, 2e, 4a, 9c, d8, 6b, dd, 36, b0
7	48, 8d, a4, 39, 3a, 5c, 38, 7a, df, 57, 00, 00, 29, 02, 09, 37	

Table 4. 6-round linear distinguisher ($p = 227$, matrix3) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 03	00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 6e
2	5d, a2, 8d, c2, 6f, b4, ab, 68, 0d, 53, 85, 42, 4b, c1, 8c, 58	5d, a2, 8d, c2, 6f, b4, ab, 68, 0d, 53, 85, 42, 4b, c1, 8c, 61
3	40, 1a, 8e, 92, 4d, 10, 67, 2c, c5, a1, 0f, 45, 6b, 3b, 3c, 33	40, 1a, 8e, 92, 4d, 10, 67, 2c, c5, a1, 0f, 45, 6b, 3b, 3c, d6
4	6a, 90, 27, 11, a4, 3b, d8, 1d, d8, 11, 62, 60, a3, 7b, 29, 00	6a, 90, 27, 11, a4, 3b, d8, 1d, d8, 11, 62, 60, a3, 7b, 29, 00
5	dd, 45, d4, 00, 12, c5, 65, 58, 17, 59, 0d, c1, 72, 8c, 28, 89	dd, 45, d4, 00, 12, c5, 65, 58, 17, 59, 0d, c1, 72, 8c, 28, 8d
6	7b, 75, e2, 85, 33, cb, 98, b1, 25, 96, 90, 3f, 44, d9, 44, 7b	5a, 4d, 1b, dc, aa, 3c, db, 29, 4c, 1c, ce, a5, 88, 60, da, 5a
7	ce, 34, c1, 15, 00, 0b, a1, 8f, 00, be, 90, 89, 1a, 9c, 2c, ca	

Table 5. 6-round linear distinguisher ($p = 227$, matrix4) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 35, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00	00, 00, 00, 02, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
2	ce, 02, 93, 17, 7d, ae, e0, af, 6e, 7c, 2c, ad, 65, d3, 9a, 26	ce, 02, 93, 17, 7d, ae, e0, af, 6e, 7c, 2c, ad, 65, d3, 9a, b6
3	e1, 49, 46, 78, 4d, c9, 0d, 9b, 08, 8e, 4a, 97, 1c, d6, ca, 09	e1, 49, 46, 78, 4d, c9, 0d, 9b, 08, 8e, 4a, 97, 1c, d6, ca, 66
4	1e, 9b, 77, e1, 23, cb, 36, c1, ba, b8, 35, ce, b3, 42, 1a, 31	1e, 9b, 77, e1, 23, cb, 36, c1, ba, b8, 35, ce, b3, 42, 1a, 04
5	dc, 04, 9c, b0, 1b, 5d, c0, 58, b7, c9, 19, 13, 73, c4, 99, 70	dc, 04, 9c, b0, 1b, 5d, c0, 58, b7, c9, 19, 13, 73, c4, 99, e0
6	79, 4a, ae, 67, 76, 09, 2c, ab, c5, 79, 65, 6c, 95, 00, 5b, 27	9e, b1, be, 0f, c9, 72, 2f, 8a, 5f, 49, 2a, 5e, e2, 00, 2f, ab
7	55, 97, 00, 3a, 0e, 04, 20, a4, a0, 53, d8, 5f, 27, 42, 00, d0	

- Matrix1:

$$X = [34, 80, 174, 199, 153, 19, 111, 1, 56, 150, 198, 119, 125, 60, 47, 78],$$

$$Y = [86, 196, 154, 178, 135, 219, 61, 194, 127, 170, 137, 43, 211, 68, 93, 4].$$

- Matrix2:

$$X = [81, 27, 2, 10, 213, 71, 94, 181, 62, 74, 192, 111, 139, 23, 54, 115],$$

$$Y = [183, 228, 32, 166, 126, 70, 190, 11, 6, 91, 187, 216, 66, 33, 145, 7].$$

- Matrix3:

$$X = [228, 65, 24, 172, 16, 124, 10, 197, 157, 27, 107, 44, 87, 84, 115, 92],$$

$$Y = [176, 173, 88, 139, 54, 208, 63, 15, 3, 86, 114, 83, 164, 155, 120, 82].$$

Table 6. 6-round linear distinguisher ($p = 233$, matrix1) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 00, 57, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00	00, 00, 00, 00, 6f, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
2	6B, A7, 52, AB, 39, B8, 7E, BC, 79, 83, 7C, 31, A6, 08, 2C, 84	6B, A7, 52, AB, 39, B8, 7E, BC, 79, 83, 7C, 31, A6, 08, 2C, be
3	DC, 41, 6F, A8, 18, 77, 8B, D1, 58, E7, AF, 42, 12, 38, 44, 53	DC, 41, 6F, A8, 18, 77, 8B, D1, 58, E7, AF, 42, 12, 38, 44, 23
4	35, E3, 37, DA, 26, 2A, E8, 86, 9B, 87, 09, 5A, 17, 3A, 51, 02	35, E3, 37, DA, 26, 2A, E8, 86, 9B, 87, 09, 5A, 17, 3A, 51, 80
5	12, 8E, D6, E7, 10, B6, 37, 22, 7E, 73, B3, 2A, 82, AE, 14, 07	12, 8E, D6, E7, 10, B6, 37, 22, 7E, 73, B3, 2A, 82, AE, 14, 08
6	12, 65, 7d, 78, 91, 4e, 00, ad, 2e, cf, 7f, 7a, 52, 2f, cc, 74	53, 25, 9e, aa, 29, 18, 00, 8e, 6d, 01, 2a, c1, 1a, 36, cb, 8f
7	00, 8d, 9c, 00, 05, 8c, 67, e6, 9e, d5, af, dd, 10, 16, 5c, 64	

Table 7. 6-round linear distinguisher ($p = 233$, matrix2) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 20, 00, 00	00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, e2, 00, 00
2	B1, 37, 8D, 2F, 24, D2, A5, 32, 19, 09, 3B, 0B, 7A, 2F, 95, 2c	B1, 37, 8D, 2F, 24, D2, A5, 32, 19, 09, 3B, 0B, 7A, 2F, 95, 97
3	5B, AF, 81, C2, 4F, 7F, D2, 35, D4, 28, 5E, 6D, 76, 8D, 1A, 5c	5B, AF, 81, C2, 4F, 7F, D2, 35, D4, 28, 5E, 6D, 76, 8D, 1A, ba
4	5F, DD, 4D, 75, 91, 8B, D5, 16, 1D, 8E, 35, 53, 8E, 6D, CF, 6f	5F, DD, 4D, 75, 91, 8B, D5, 16, 1D, 8E, 35, 53, 8E, 6D, CF, 85
5	DE, 03, 58, 01, 5B, 42, A9, 3F, 8C, AD, 7F, 74, 56, 5C, 3E, 00	DE, 03, 58, 01, 5B, 42, A9, 3F, 8C, AD, 7F, 74, 56, 5C, 3E, 00
6	bf, 96, 9d, 3a, 41, 9c, 40, 9f, df, b0, df, 1a, 1b, cf, e4, 0a	35, 96, a1, 2f, 82, 53, a6, 9b, b0, b7, b0, e8, 86, 01, 16, 1e
7	86, 39, 00, 4d, 7f, ca, cd, a6, 00, 5d, ab, 4c, 01, d1, 49, 15, 00	

Table 8. 6-round linear distinguisher ($p = 233$, matrix3) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 00, 00, 00, 00, 5a, 00, 00, 00, 00, 00, 00, 00, 00	00, 00, 00, 00, 00, 00, 00, d7, 00, 00, 00, 00, 00, 00, 00, 00
2	24, DC, 7D, A3, BB, 09, D6, E8, 10, 26, 95, D5, 70, 66, 4E, 10	24, DC, 7D, A3, BB, 09, D6, E8, 10, 26, 95, D5, 70, 66, 4E, 02
3	76, 56, E4, 9A, 60, 78, 69, 00, 3A, 4B, 96, 6A, 54, 83, 49, 83	76, 56, E4, 9A, 60, 78, 69, 00, 3A, 4B, 96, 6A, 54, 83, 49, 3e
4	21, 85, C2, 6A, B9, B6, 1B, AD, E0, D3, 6A, A6, 55, C5, 65, be	21, 85, C2, 6A, B9, B6, 1B, AD, E0, D3, 6A, A6, 55, C5, 65, e0
5	C9, A5, 78, E0, E4, 3B, 71, 0B, A5, 33, 73, 86, C2, 40, 67, 00	C9, A5, 78, E0, E4, 3B, 71, 0B, A5, 33, 73, 86, C2, 40, 67, 00
6	15, 12, 15, 31, bb, 1f, ae, 17, dd, 47, 94, 42, b2, 4f, 16, d5	cc, 93, 9f, 82, 35, 06, 97, db, 72, 09, cf, 51, c1, dc, db, 0a
7	a8, 3a, d8, 43, c6, ba, 9e, 25, a4, 36, a1, 00, 32, 88, 00, 4c	

Table 9. 6-round linear distinguisher ($p = 233$, matrix4) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 9d, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00	00, 00, 00, a1, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
2	4B, 08, E4, 88, 12, 19, E7, 3A, 8F, A7, B4, B6, 8F, 15, 36, 69	4B, 08, E4, 88, 12, 19, E7, 3A, 8F, A7, B4, B6, 8F, 15, 36, 12
3	56, 24, 77, 71, 91, C8, 4E, 6F, 20, 25, 8B, 7C, 6D, 6D, AE, 00	56, 24, 77, 71, 91, C8, 4E, 6F, 20, 25, 8B, 7C, 6D, 6D, AE, 00
4	06, CC, D9, 06, 82, 49, D0, 15, 08, CA, AA, 13, 80, 51, 44, 3f	06, CC, D9, 06, 82, 49, D0, 15, 08, CA, AA, 13, 80, 51, 44, b7
5	2D, 05, 22, 78, 30, CD, 8B, 5E, 95, 17, A7, 0A, 70, C9, 45, 0f	2D, 05, 22, 78, 30, CD, 8B, 5E, 95, 17, A7, 0A, 70, C9, 45, c9
6	b0, 9f, 1f, 79, 8d, 38, 3b, 58, 71, a7, 63, 67, 16, 52, e6, b2	39, 9b, 3a, 89, c5, d2, 52, c0, 77, 50, ba, 7d, 39, de, d5, c1
7	16, 1b, 28, e7, 6b, a3, 3d, 43, 80, ac, 0b, c9, 74, 00, 51, 00	

- Matrix4:

$$X = [38, 8, 88, 226, 159, 188, 165, 103, 217, 137, 129, 140, 143, 157, 231, 110],$$

$$Y = [151, 179, 200, 4, 198, 114, 187, 94, 116, 199, 168, 219, 189, 81, 180, 191].$$

$p = 239$.

- Matrix1:

$$X = [57, 46, 202, 32, 190, 104, 54, 174, 63, 114, 120, 27, 186, 35, 160, 115],$$

$$Y = [42, 91, 6, 44, 90, 131, 201, 164, 62, 39, 48, 70, 69, 127, 139, 158].$$

- Matrix2:

$$X = [181, 218, 128, 122, 134, 236, 96, 195, 155, 156, 68, 15, 23, 217, 28, 85],$$

$$Y = [142, 42, 130, 215, 135, 148, 93, 50, 73, 174, 186, 7, 198, 150, 210, 30].$$

- Matrix3:

$$X = [135, 34, 176, 227, 29, 22, 37, 218, 224, 119, 60, 75, 183, 214, 171, 88],$$

$$Y = [54, 225, 173, 85, 77, 137, 199, 203, 230, 27, 174, 65, 13, 131, 4, 32].$$

- Matrix4:

$$X = [86, 157, 220, 85, 59, 227, 154, 206, 50, 84, 23, 125, 64, 105, 134, 103],$$

$$Y = [217, 40, 25, 43, 193, 49, 1, 160, 46, 94, 186, 97, 108, 161, 131, 37].$$

Table 10. 6-round linear distinguisher ($p = 239$, matrix1) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 00, 24, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00	00, 00, 00, 00, b5, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
2	57, 0E, 90, EB, 72, 4A, 3C, 57, AA, 1A, 19, 0C, 60, 16, 8E, 6b	57, 0E, 90, EB, 72, 4A, 3C, 57, AA, 1A, 19, 0C, 60, 16, 8E, 47
3	15, AD, 02, AB, 69, 5D, 56, 35, 81, 88, CB, C7, 8C, D4, D5, 58	15, AD, 02, AB, 69, 5D, 56, 35, 81, 88, CB, C7, 8C, D4, D5, 6c
4	CD, 85, E9, 6A, 3E, B5, 86, A0, 9B, 33, 03, 52, C4, 18, 04, d3	CD, 85, E9, 6A, 3E, B5, 86, A0, 9B, 33, 03, 52, C4, 18, 04, 6e
5	5D, EC, 9F, E2, 24, C2, C5, 81, 3D, 9B, 67, 72, 95, B3, A9, e5	5D, EC, 9F, E2, 24, C2, C5, 81, 3D, 9B, 67, 72, 95, B3, A9, 34
6	d5, 44, 92, c1, af, af, 33, ad, 4c, 1c, 00, 72, 00, 9d, 9e, 79	8b, 3f, b8, b1, 55, 1b, bc, 09, 8e, 43, 00, 19, 00, 4d, 1d, 4a
7	b2, 27, d0, 00, 9d, e1, 79, 00, 96, 1f, 53, a4, bc, 38, 2a, 5e	

Table 11. 6-round linear distinguisher ($p = 239$, matrix2) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, bf, 00, 00	00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 69, 00, 00
2	B1, 32, 93, BB, 4E, 30, C8, 79, 29, 64, D1, 80, 1B, 08, E9, 6f	B1, 32, 93, BB, 4E, 30, C8, 79, 29, 64, D1, 80, 1B, 08, E9, 44
3	70, 7F, 0D, E3, 8E, E9, 08, 94, 20, 10, 82, 6C, 9B, EB, 7E, c7	70, 7F, 0D, E3, 8E, E9, 08, 94, 20, 10, 82, 6C, 9B, EB, 7E, 80
4	CD, 66, 0E, B9, 30, 7E, EB, 11, 69, 64, 30, A6, E0, A4, 5B, 8b	CD, 66, 0E, B9, 30, 7E, EB, 11, 69, 64, 30, A6, E0, A4, 5B, 1c
5	A4, 1A, EB, 9A, DD, BB, 36, 8C, 1B, 69, 3A, 47, 0D, A9, 6D, 23	A4, 1A, EB, 9A, DD, BB, 36, 8C, 1B, 69, 3A, 47, 0D, A9, 6D, 94
6	3c, 09, b9, 16, 00, 71, 9c, 72, 00, 26, a0, a1, 49, ee, b6, 1b	b8, d2, 59, 65, 00, e4, 3c, 3d, 00, 6e, 56, 59, 02, ae, ce, 98
7	8a, d5, 00, 0b, cc, 08, de, 00, 44, a2, e6, 57, 72, 9b, 65, cd	

Table 12. 6-round linear distinguisher ($p = 239$, matrix3) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	01, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00	12, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
2	DD, 57, 1F, D8, 27, A8, 51, EE, A6, A5, 81, BE, 7D, 01, 89, 69	DD, 57, 1F, D8, 27, A8, 51, EE, A6, A5, 81, BE, 7D, 01, 89, 40
3	ED, DA, 20, 86, 63, 16, 90, 24, 50, 84, 03, CA, 77, AF, 16, 57	ED, DA, 20, 86, 63, 16, 90, 24, 50, 84, 03, CA, 77, AF, 16, 1d
4	82, 99, E4, 40, A8, ED, 34, CE, BF, 25, 7F, 07, ED, 81, C3, 00	82, 99, E4, 40, A8, ED, 34, CE, BF, 25, 7F, 07, ED, 81, C3, 00
5	60, 70, 57, 10, 24, D4, 66, BB, 5B, B4, 3B, C7, B3, 28, C8, 18	60, 70, 57, 10, 24, D4, 66, BB, 5B, B4, 3B, C7, B3, 28, C8, 02
6	8d, b0, a7, a2, 00, e7, 37, 17, b9, 29, d2, 96, 7b, 10, 31, e3	40, 94, 7a, 07, 00, 7a, 5f, 53, 32, 04, a6, 67, 01, 4e, 22, 71
7	44, b2, 00, 32, 1c, 9d, a7, 2e, 01, 00, d3, 79, 24, 6c, 29, 95	

Table 13. 6-round linear distinguisher ($p = 239$, matrix4) (hexadecimal representation).

Round	masks before S-boxes $\Gamma_{in,i}$	masks after S-boxes $\Gamma_{out,i}$
1	00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 36, 00, 00, 00, 00	00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, ce, 00, 00, 00, 00
2	22, CA, 76, 18, 88, 32, BA, CA, EA, 2B, 9C, 40, 1E, 74, B2, 61	22, CA, 76, 18, 88, 32, BA, CA, EA, 2B, 9C, 40, 1E, 74, B2, d6
3	A6, 15, 3A, 9E, B6, A7, 9B, 07, CC, BB, E7, 1D, 2D, 1C, DD, da	A6, 15, 3A, 9E, B6, A7, 9B, 07, CC, BB, E7, 1D, 2D, 1C, DD, ce
4	ED, 4B, 9C, BF, 33, 39, 09, 97, DA, 02, CC, 6A, 14, 98, 51, 77	ED, 4B, 9C, BF, 33, 39, 09, 97, DA, 02, CC, 6A, 14, 98, 51, 98
5	C4, BE, 23, 24, 21, 0E, E5, 6E, 93, 92, ED, E1, 25, E7, 15, 85	C4, BE, 23, 24, 21, 0E, E5, 6E, 93, 92, ED, E1, 25, E7, 15, 25
6	08, b9, b1, 55, 33, 1b, a0, a6, 92, 72, 30, 00, 58, cd, 30, bc	bd, 2c, 31, 38, bc, 72, 4c, c7, 51, 3d, b1, 00, 8a, 16, b1, 24
7	17, d4, 00, 6e, 42, 40, 7f, 00, ce, eb, 9c, 25, 56, a6, cd, 50	

References

1. Albrecht, M.R., et al.: Algebraic cryptanalysis of STARK-friendly designs: application to MARVELlous and MiMC. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 371–397. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_13
2. Albrecht, M.R., et al.: Feistel structures for MPC, and more. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019. LNCS, vol. 11736, pp. 151–171. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29962-0_8
3. Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 191–219. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_7
4. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. IACR Trans. Symmetric Cryptol. **2020**(3), 1–45 (2020)
5. Baignères, T., Stern, J., Vaudenay, S.: Linear cryptanalysis of non binary ciphers. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 184–211. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77360-3_13
6. Bar-On, A., Dunkelman, O., Keller, N., Weizman, A.: DLCT: a new tool for differential-linear cryptanalysis. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 313–342. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_11
7. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 825–885

8. Beierle, C., et al.: Improved differential-linear attacks with applications to ARX ciphers. J. Cryptol. **35**(4), 29 (2022)
9. Beyne, T., et al.: Out of oddity – new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 299–328. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_11
10. Biham, E., Dunkelman, O., Keller, N.: Enhancing differential-linear cryptanalysis. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 254–266. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36178-2_16
11. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991)
12. Biryukov, A., De Cannière, C., Quisquater, M.: On multiple linear approximations. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 1–22. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_1
13. Bleichenbacher, D.: On the generation of DSA one-time keys. In: Presentation at Cryptography Research Inc., San Francisco (2007)
14. Blondeau, C., Leander, G., Nyberg, K.: Differential-linear cryptanalysis revisited. J. Cryptol. **30**(3), 859–888 (2017)
15. Blondeau, C., Nyberg, K.: New links between differential and linear cryptanalysis. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 388–404. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_24
16. Bogdanov, A., et al.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_31
17. Bogdanov, A., Rijmen, V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Des. Codes Cryptogr. **70**(3), 369–383 (2014)
18. Chabaud, F., Vaudenay, S.: Links between differential and linear cryptanalysis. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 356–365. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0053450
19. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Improving the time complexity of Matsui's linear cryptanalysis. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 77–88. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76788-6_7
20. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC, pp. 151–158. ACM (1971)
21. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**(90), 297–301 (1965)
22. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. ISC, Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-662-04722-4
23. Davis, P.J.: Circulant Matrices. American Mathematical Society (2013)
24. Dobraunig, C., Grassi, L., Guinet, A., Kuijsters, D.: Ciminion: symmetric encryption based on Toffoli-Gates over large finite fields. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 3–34. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_1
25. Eichlseder, M., et al.: An algebraic attack on ciphers with low-degree round functions: application to full MiMC. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 477–506. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_16

26. Flórez-Gutiérrez, A.: Optimising linear key recovery attacks with affine Walsh transform pruning. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. LNCS, vol. 13794, pp. 447–476. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22972-5_16
27. Flórez-Gutiérrez, A., Naya-Plasencia, M.: Improving key-recovery in linear attacks: application to 28-round PRESENT. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 221–249. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_9
28. Grassi, L., Hao, Y., Rechberger, C., Schofnegger, M., Walch, R., Wang, Q.: A new Feistel approach meets fluid-SPN: griffin for zero-knowledge applications. IACR Cryptology ePrint Archive, p. 403 (2022)
29. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: Reinforced concrete: a fast hash function for verifiable computation. In: CCS, pp. 1323–1335. ACM (2022)
30. Grassi, L., Lüftenegger, R., Rechberger, C., Rotaru, D., Schofnegger, M.: On a generalization of substitution-permutation networks: the HADES design strategy. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 674–704. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_23
31. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON block cipher family. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 161–185. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_8
32. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 17–25. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_3
33. Liu, Z., Gu, D., Zhang, J., Li, W.: Differential-multiple linear cryptanalysis. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 35–49. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16342-5_3
34. Lu, J.: A methodology for differential-linear cryptanalysis and its applications. Des. Codes Cryptogr. **77**(1), 11–48 (2015)
35. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_33
36. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C.-K., Yung, M., Lin, D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34704-7_5
37. Sun, S., et al.: Analysis of AES, SKINNY, and others with constraint programming. IACR Trans. Symmetric Cryptol. **2017**(1), 281–306 (2017)
38. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_9

A New Correlation Cube Attack Based on Division Property

Cheng Che[1,2] and Tian Tian[1(✉)]

[1] Information Engineering University, Zhengzhou 450001, China
tiantian_d@126.com
[2] State Key Laboratory of Cryptology, P. O. Box 5159, Beijing 100878, China

Abstract. Correlation cube attacks were proposed by Liu et al. at EUROCRYPT 2018, which targeted a modern symmetric-key cryptosystem based on nonlinear feedback shift registers. The main idea of correlation cube attacks lies in recovering the secret key by exploiting conditional correlation properties between the superpoly of a cube and a specific set of low-degree polynomials called a basis. In this paper, we propose a new correlation cube attack based on the division property. The new attack expresses a given superpoly p as $fg \oplus h$ and calculates correlation probabilities for all possible f, where f only involves key variables. This novel idea breaks the restriction on the basis used in the original attack and provides more choices to find good correlation probabilities, thus making the correlation cube attack much more powerful. Besides, the first application of the division property to correlation cube attacks is given, which aided by MILP modeling techniques facilitates the search for desirable cubes. As illustrations, we apply the new correlation cube attack to Trivium. For 844-round Trivium, we can recover about 4-bit key information on average with the time complexity 2^{45}, which could be fully verified by experiments. This is so far the best key recovery attack for 844-round Trivium.

Keywords: Cube Attack · Division Property · Stream cipher · Trivium

1 Introduction

The cube attack is a powerful cryptanalytic technique against symmetric cryptosystems proposed by Dinur and Shamir at Eurocrypt 2009 [1]. The attack process of cube attacks includes preprocessing and online phases. In the preprocessing phase, effective cubes are obtained by search, and the corresponding superpolies are recovered. In the online phase, all possible values of the cube variables are assigned, and then the encryption oracle is queried and summed to obtain the value of the superpolies under the real key. Finally, information about the secret key can be revealed by solving the system of equations. It can

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 53–71, 2023.
https://doi.org/10.1007/978-3-031-35486-1_3

be seen that the key to a successful cube attack is to search for good cubes and recover superpolies.

Recovering Superpolies. In traditional cube attacks [1–3], attackers regard the cryptographic algorithm as a black box and experimentally search for linear or quadratic superpolies. In [4], the conventional bit-based division property (CBDP) was first introduced to cube attacks on stream ciphers. For a given cube set, Todo et al. established and solved the MILP model to identify the secret variables not involved in the superpoly. In order to improve the effectiveness of the model, some new techniques were given in [5,6]. However, neither experimental cube attacks nor CBDP-based cube attacks can accurately recover superpolies. In [7,8], Ye and Tian preliminarily solved this problem. Several superpolies are recovered by recursively-expressing [7] and algebraic [8] methods, respectively, but both methods have limitations. At Eurocrypt 2020 [9], Hao et al. proposed the three-subset division property without unknown subsets to recover the exact superpolies, thus successfully solving this problem. Meanwhile, Hu et al. in [10] proposed a monomial prediction technique based on the propagation of monomials. Moreover, they established the equivalence between the technique and the three-subset division property without unknown subsets. Later, Hu et al. [11] combined the monomial prediction technique with the recursively-expressing method in [7] and presented a nested framework for recovering the ANFs of massive superpolies. Very recently, He et al. in [12] introduced two new techniques on the nested framework, which allows attackers to recover the superpolies that contain more than 500 million terms.

Searching for Cubes. As the method of recovering superpolies becomes more and more effective, another problem of cube attacks, i.e., how to select good cubes to make a cube attack more effective, is attracting more and more attention. Since a system of linear equations is quite easy to solve, linear superpolies are most useful in a cube attack. At Asiacrypt 2021, based on the greedy strategy and the degree evaluation method, Ye et al. in [13] proposed a new algorithm for constructing cubes associated with linear superpolies, leading to a practical attack against 805-round Trivium. Besides linear superpolies, some kinds of nonlinear superpolies also useful. Later, Sun in [14] pointed out that the class of nonlinear superpolies with independent secret variables could be easily exploited to recover key information simply by the linearization technique, where an independent secret variable is a variable involving in a superpoly but does not appear in any nonlinear term of the superpoly. Consequently, Sun proposed a heuristic method in [14] to reject cubes without independent secret variables from a preset of candidate cubes. Using the heuristic algorithm, the author of [14] recovered a balanced superpoly for 843-round Trivium and presented practical attacks against 806- and 808-round Trivium. Recently, Che et al. considered that the linear term is necessary for superpolies to contain independent secret variables. Thus, a new framework was proposed in [15], and practical attacks against 815- and 820-round Trivium were given. How to use other forms of balanced nonlinear superpolies in cube attacks remains open, especially massive superpolies.

Cube Attack Variants. With the improvement of cube attacks, some enlightening attack ideas are introduced and proposed. In [16], Dinur and Shamir proposed dynamic cube attacks. The main idea of dynamic cube attacks is to simplify the superpolies by annihilating some expressions about secret variables and public variables. Then, attackers can determine the correctness of the guessed values by detecting the nonrandomness of the high-order differences of the output bits. Similar to the dynamic cube attack, the conditional cube attack [17] also obtains a distinguisher by adding the conditions of the key and IV. In order to convert this kind of attack from a weak-key distinguisher to a key recovery attack, Liu et al. proposed the correlation cube attack in [18]. In the correlation cube attack, the attacker recovers the secret key through the conditional correlation properties between the superpoly of a cube and the annihilated conditions. The set of annihilation conditions that can make a superpoly become zero is called a basis.

Motivation: Although we can recover a massive superpoly, it is unclear how a superpoly with millions of terms can be used in an attack. In [11], Hu et al. recovered two superpolies for 844-round Trivium, and the two superpolies are roughly balanced through experimental tests. When attacking the 844-round Trivium, Hu et al. proposed that two linear equations could be obtained when guessing the values of 78 variables. However, in many cases, two equations are the same, or even both are 0, which yields that the values of two unknown variables cannot be solved. In addition, compared with querying the encryption oracle that calls 844-round Trivium, there is a doubt whether the complexity of assigning 78 variables to a massive superpoly can be ignored. Therefore, before proceeding to recover even more massive superpolies, we need to think over its significance in the security evaluation.

Our Contribution: In this paper, we propose a new correlation cube attack. In the new attack, we express the superpoly p as $fg \oplus h$, where f only involves secret bits. We look for the cube where h is biased. Then, we can use the correlation between p and f to obtain the secret key information.

The critical step to correlation cube attacks is to search for good cubes. Our cube search is inspired by the idea of basis test in [18]. We introduce the division property into the basis test. Based on the three-subset division property without unknown subset, we propose a modeling method for polynomials and give the basis test based on the division property. Therefore, we could more accurately evaluate whether the superpoly is a zero constant after adding some conditions, which significantly improves the power of the search.

As an illustration, we apply the attack to the well-known stream cipher Trivium and obtain the best-known key recovery results for 844-round Trivium. For 844-round Trivium, we could obtain about 4-bit key information at 2^{45} online computation complexity. Hence, the complexity of recovering the secret key could be reduced to about 2^{76}. The attack solves the problem that the superpolies of high-round Trivium are too complex to use directly. As a comparison, we summarize full key-recovery attacks against the round-reduced Trivium in Table 1.

Table 1. A summary of key-recovery attacks on Trivium

Rounds	#Cube	Cube size	Time complexity	Ref
≤820	-	-	Practical	[1,2,13–15]
≤843*	-	-	2^{75}–$2^{79.6}$	[4–11,14,18]
844	2	52–53	2^{78}	[11]
844	28	37–38	2^{76}	Sect. 5.2
845	2	54–55	2^{78}	[11]
846	6	51–54	2^{79}	[12]
847	2	52–53	2^{79}	[12]
848	1	52	2^{79}	[12]

$*$: The highest round of Trivium with balanced superpoly is 843.

Related Work. Similar to correlation cube attacks in [18], the new correlation cube attacks recover the key by exploiting some probabilistic equations. The main difference between the two attacks is the perspective of viewing the superpoly p. In the original attack, the superpoly p is decomposed into $\bigoplus_{i=1}^{u} f_i \cdot g_i$, and u is required not to be large. However, as the number of attack rounds increases, it is difficult to bound u. The new attack describes the superpoly p as $fg \oplus h$ and looks for h with high bias. Then, f_i in the original attack can be regarded as f in the new attack. Therefore, the new correlation cube attacks can be considered a generalization of the original correlation cube attacks. Moreover, due to the new perspective, we can get more and better correlations, which is shown in the following two points.

- The original attack only verified the correlation with the elements in the basis. In the new perspective, we will test all possible f, such as low-degree polynomials appearing in the low round. The experimental data show that more correlations can be obtained.
- The original attack successfully attacked 835-round Trivium. However, when attacking 840 or more rounds, we find that the algorithm is not always efficient: very few cubes pass the basis test. In comparison, using the same candidate cube set, the new method can find more useful cubes.

Organization. This paper is organized as follows. Section 2 introduces necessary notations and some preliminaries. The details of the new correlation cube attacks are described in Sect. 3 and Sect. 4, and its applications to Trivium are in Sect. 5. Finally, conclusions are drawn in Sect. 6. The source codes of the algorithm, including some experimental data, were released at https://github.com/LLuckyRabbit/NewCorrelationCubeAttack.

2 Preliminaries

In this section, we introduce some related concepts and definitions.

2.1 Boolean Functions

Let $\mathbb{F}_2$ be the binary field and $\mathbb{F}_2^n$ be an n-dimensional vector space over $\mathbb{F}_2$. An n-variable Boolean function is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. A Boolean function f can be uniquely represented as a multivariable polynomial over $\mathbb{F}_2$,

$$f(x_1, x_2, \ldots, x_n) = \bigoplus_{\boldsymbol{c}=(c_1,c_2,\ldots,c_n)\in\mathbb{F}_2^n} a_{\boldsymbol{c}} \prod_{i=1}^{n} x_i^{c_i},\ a_{\boldsymbol{c}} \in \mathbb{F}_2,$$

which is called the algebraic normal form (ANF) of f, and $\prod_{i=1}^{n} x_i^{c_i}$ is called a term of f. The algebraic degree of f is denoted by $\deg(f)$ and defined as $\max\{wt(\boldsymbol{c})|a_{\boldsymbol{c}} \neq 0\}$, where $wt(\boldsymbol{c})$ is the Hamming Weight of $\boldsymbol{c}$.

In particular, given a Boolean function f, we can express f in the form of $\bigoplus_{i=1}^{u} f_i \cdot g_i$, and call this form the decomposition of f on the basis $G = \{g_1, g_2, \ldots, g_u\}$. It is clear that the basis of f is not unique.

2.2 Cube Attacks

Given a cryptographic algorithm, its output bit can be regarded as a Boolean function in the secret variables $\boldsymbol{k} = (k_1, k_2, \ldots, k_n)$ and the public variables $\boldsymbol{v} = (v_1, v_2 \ldots, v_m)$, expressed as $f(\boldsymbol{k}, \boldsymbol{v})$. For a randomly chosen set $I = \{v_{i_1}, \ldots, v_{i_d}\}$, $f(\boldsymbol{k}, \boldsymbol{v})$ can be represented uniquely as

$$f(\boldsymbol{k}, \boldsymbol{v}) = t_I \cdot p_I(\boldsymbol{k}, \boldsymbol{v}) \oplus q_I(\boldsymbol{k}, \boldsymbol{v}),$$

where $t_I = v_{i_1} \cdots v_{i_d}$, and q_I misses at least one variable in I. The basic idea of a cube attack is that the summation of the 2^d functions derived from $f(\boldsymbol{k}, \boldsymbol{v})$ by assigning all the possible values to variables in I equals $p_I(\boldsymbol{k}, \boldsymbol{v})$, that is,

$$\bigoplus_{(v_{i_1}, v_{i_2}, \ldots, v_{i_d})\in\mathbb{F}_2^d} f(\boldsymbol{k}, \boldsymbol{v}) = p_I(\boldsymbol{k}, \boldsymbol{v}).$$

The public variables in I are called cube variables, and the polynomial $p_I(\boldsymbol{k}, \boldsymbol{v})$ is called the superpoly of the cube I. Obviously, $p_I(\boldsymbol{k}, \boldsymbol{v})$ is much simpler than $f(\boldsymbol{k}, \boldsymbol{v})$ and is expected to be obtained. Therefore, when an attacker recovers a certain number of superpolies, he could build a system of equations on secret variables $\boldsymbol{k}$ by inquiring about the values of all the superpolies. Then some information about the secret variables can be achieved.

2.3 Numeric Mapping

The numeric mapping was first introduced at CRYPTO 2017 [19], mainly used to evaluate the degree of NFSR-based cryptosystems. Let

$$f(x_1, x_2, \ldots, x_n) = \bigoplus_{\boldsymbol{c}=(c_1,c_2,\ldots,c_n)\in\mathbb{F}_2^n} a_{\boldsymbol{c}} \prod_{i=1}^{n} x_i^{c_i},\ a_{\boldsymbol{c}} \in \mathbb{F}_2,$$

be a Boolean function on n variables. The numeric mapping, denoted by DEG, is defined as

$$\begin{aligned}\mathrm{DEG}: \mathbb{B}_n \times \mathbb{Z}^n &\to \mathbb{Z}\\ (f, D) &\mapsto \max_{a_c \neq 0} \sum_{i=1}^{n} c_i d_i,\end{aligned}$$

where $D = (d_1, d_2, \ldots, d_n)$. Let $d_i (1 \leq i \leq n)$ be the degree of Boolean function g_i, and then $\mathrm{DEG}(f, D)$ is the degree evaluation of composite function $f(g_1, \ldots, g_n)$. Based on the numeric mapping, the algebraic degree of output bits and internal states of a cryptographic algorithm can be evaluated.

2.4 Correlation Cube Attacks

Correlation cube attacks were proposed by Liu et al. [18]. In the correlation cube attack, the attacker can establish deterministic or probabilistic equations about the secret key by using the correlation properties between the superpoly p_I and a set of low-degree decomposition basis. Correlation cube attacks mainly include the following three steps.

a. Search for a cube I such that its superpoly p_I can be decomposed into $\bigoplus_{g \in G} f_g \cdot g$, where g is a low-degree polynomial on key variables, and G is called a low-degree decomposition basis.
b. For $g \in G$, the correlation properties between g and p_I is calculated, that is, the probability $\Pr(g = p_I)$ is calculated.
c. Query the encryption oracle and calculate the value of p_I, denoted as a. Then we have $g = a$ is established with probability $\Pr(g = p_I)$.

Compared with cube attacks, it seems that correlation cube attacks can attack ciphers with higher rounds with cubes of smaller dimensions.

2.5 The Division Property

The division property was proposed at Eurocrypt 2015 [20], which is a new technique to find integral characteristics. Subsequently, the conventional bit-based division property is introduced in [21]. At the same time, the authors of [21] proposed the three-subset division property whose definition is as follows.

Definition 1 (Three-subset division property [21]). *Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^n$. When the multiset $\mathbb{X}$ has the three-subset division property $D^{1^n}_{\mathbb{K},\mathbb{L}}$, it fulfills the following conditions:*

$$\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = \begin{cases} \textit{unknown} & \textit{if there exists } \boldsymbol{\alpha} \textit{ in } \mathbb{K} \textit{ s.t. } \boldsymbol{u} \succeq \boldsymbol{\alpha},\\ 1 & \textit{else if there exists } \boldsymbol{\beta} \textit{ in } \mathbb{L} \textit{ s.t. } \boldsymbol{u} = \boldsymbol{\beta},\\ 0 & \textit{otherwise,}\end{cases}$$

where $\boldsymbol{u} \succeq \boldsymbol{\alpha}$ if and only if $u_i \geq k_i$ for all i and $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=1}^{n} x_i^{u_i}$.

The propagation of division property can be used to determine whether an integral characteristic exists. However, the computation becomes unacceptable as the number of attack rounds increases. To solve this problem, Xiang et al. in [22] first introduced the concept of division trails, a set of ordered vectors describing the propagation of division property. Based on the concept of division trails, Xiang et al. proposed to use Mixed Integer Linear Programming (MILP) to model the conventional bit-based division property and transform the problem into solving MILP problems. In order to model the three-subset division property directly in cube attacks, Hao et al. proposed the three-subset division property without unknown subset in [9].

Definition 2 (Three-subset division property without unknown subset [9]). *Let $\mathbb{X}$ and $\tilde{\mathbb{L}}$ be two multisets whose elements take a value of $\mathbb{F}_2^n$. When the multiset $\mathbb{X}$ has the three-subset division property without unknown subset $\Gamma_{\tilde{\mathbb{L}}}^{1^n}$, it fulfills the following conditions:*

$$\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = \begin{cases} 1 & \textit{if there are odd-number of } \boldsymbol{u}\textit{'s in } \tilde{\mathbb{L}}, \\ 0 & \textit{otherwise,} \end{cases}$$

where $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=1}^{n} x_i^{u_i}$.

Definition 2 is a great improvement on the three-subset division property used in cube attacks against stream ciphers, which abandon the unknown sets in the division property, but only describes the propagation of $\mathbb{L}$ set and zero set, so as to obtain the MILP models for recovering superpolies. The propagation rule of the three-subset division property without unknown subset is shown for three basic operations: "copy", "and" and "xor" in [9].

Rule 1 (copy): Let $a \xrightarrow{\text{copy}} (b_1, b_2)$ be a division trail of copy. The following inequalities describe the propagation of the three-subset division property without unknown subset for copy.

$$\begin{cases} \mathcal{M}.var \leftarrow a, b_1, b_2 \textit{ as binary} \\ \mathcal{M}.con \leftarrow b_1 + b_2 \geq a \\ \mathcal{M}.con \leftarrow a \geq b_1 \\ \mathcal{M}.con \leftarrow a \geq b_2 \end{cases}$$

When the "or" operation is supported in the MILP solver, such as Gurobi, we can simply write $\mathcal{M}.con \leftarrow a = b_1 \vee b_2$.

Rule 2 (and): Let $(a_1, a_2, \ldots, a_m) \xrightarrow{\text{and}} b$ be a division trail of and. The following inequalities describe the propagation of the three-subset division property without unknown subset for and.

$$\begin{cases} \mathcal{M}.var \leftarrow a_1, a_2, \ldots, a_m, b \textit{ as binary} \\ \mathcal{M}.con \leftarrow b = a_i \textit{ for all } i \in \{1, 2, \ldots, m\} \end{cases}$$

Rule 3 (xor): Let $(a_1, a_2, \ldots, a_m) \xrightarrow{\text{xor}} b$ be a division trail of xor. The following inequalities describe the propagation of the three-subset division property without unknown subset for xor.

$$\begin{cases} \mathcal{M}.var \leftarrow a_1, a_2, \ldots, a_m, b \text{ } as \text{ } binary \\ \mathcal{M}.con \leftarrow b = a_1 + a_2 + \cdots + a_m \end{cases}$$

In the new model, each division trail represents a term in the superpoly, so the superpoly can be completely recovered by solving all feasible solutions in the model.

3 A Simple Example of the New Correlation Cube Attacks

As superpolies become increasingly massive, instead of thinking about how to recover more massive superpolies, we should think more about how to use these superpolies to carry out an effective attack. In the new correlation cube attack, by exploiting correlations between a superpoly and some key expressions, we could convert a class of superpolies into a system of simple key equations avoiding the recovery of the exact ANFs which might be very large.

Our idea is inspired by polynomials of the form $p = fg$. For such polynomials, we can regard g as a black box and guess the value of f by changing the value of g several times. Specifically, if p is constant 0 under multiple tests, it is considered that f is likely to be 0. Similarly, if p is not constant 0, then f must be 1. However, when attacking a well-designed cipher, it is difficult to have such a good form of superpoly. Therefore, we expand our ideas. In new attacks, we look for cubes whose superpolies can be expressed as

$$p = fg \oplus h,$$

where f only involve secret bits and h has a high bias. We can successfully search for superpolies of this form, which can also be used in attacks.

To demonstrate the idea of the attack, we consider a polynomial P which is a function of the three polynomials P_1, P_2, and P_3:

$$P = P_1 P_2 \oplus P_3,$$

where P_1, P_2, and P_3 are polynomials over five secret variables x_1, x_2, x_3, x_4, x_5 and five public variables v_1, v_2, v_3, v_4, v_5:

$$\begin{aligned} &P_1 = x_1x_2 \oplus x_3 \oplus x_4, \\ &P_2 = \text{arbitrary dense polynomial in the 10 variables}, \\ &P_3 = v_2x_2x_3x_4x_5 \oplus v_3x_1x_4 \oplus v_4x_2x_4 \oplus v_5x_3x_5 \oplus x_2x_3x_4 \oplus x_1x_2 \oplus v_3. \end{aligned}$$

Since P_2 is unrestricted, P is likely to behave randomly and it seems to be immune to cube attacks. However, we observe that P_3 is biased to zero under some conditions. When $v_3 = v_4 = v_5 = 0$, v_1 and v_2 take all possible combinations of 0/1, and P_3 has zero constant value for more than 70% secret variables. Therefore, for 70% secret variables, we have $P = P_1P_2$ under the setting of $v_3 = v_4 = v_5 = 0$. So, for this example, we can guess the value of P_1 from the value of P.

However, it is hard to evaluate whether h has a high bias. If p is random in the attack, that is, not constant after many tests, it is difficult to have a beneficial correlation because it is unclear whether this randomness comes from $f \cdot g$ or h. Therefore, we prefer the case where p is a constant, which can only occur in the following two cases. In the attack, the key is restricted, so the value of f is specific. If f is 0, the superpoly degenerates to h. If f is 1, the superpoly is $g + h$. Then, for both cases, not only do we know that h has a high bias, but we can also guess the specific value of f through the conditional probability calculated experimentally.

4 New Correlation Cube Attacks

In this section, we propose a new correlation cube attack based on the division property. Similar to the correlation cube attacks in [18], the attack evaluates the correlations between the superpoly and some low-degree polynomials, and recovers the key by solving systems of probabilistic equations. In the following, we will give the details of the preprocessing phase.

We are inspired by the method of finding cubes with low-degree decomposition basis in the correlation cube attack in [18]. In [18], Liu et al. can find some cubes whose superpolies p can be decomposed into $\bigoplus_{i=1}^{u} f_i \cdot g_i$, where $g_1, \ldots, g_u$ are some low-degree polynomials about key variables. So, we can naturally regard $\bigoplus_{i=2}^{u} f_i \cdot g_i$ as h, then

$$p = f_1 \cdot g_1 \oplus \bigoplus_{i=2}^{u} f_i \cdot g_i \triangleq f \cdot g \oplus h.$$

Since the key is fixed in each attack, h is actually $\bigoplus_{f_i=1, i\neq 1} g_i$, that is, h is the sum of a few polynomials. We can think that g is a complex polynomial in high-round cryptographic algorithms, such as 844-round Trivium. In particular, when g is the product of many expressions, g will have a high bias. Since the sum of the few biased polynomials is biased, we find a good cube. Therefore, we can search for desirable cubes by finding the basis in [18].

Basis Test. Since the output bits of the cryptographic algorithm are generated iteratively, the coefficients of the maximum degree terms of the cube variables in the previous N_0 rounds of state bits are likely a basis. In [18], the authors recorded these coefficients of the previous N_0 rounds and annihilated them to simplify the ANFs of output bits. Then, whether the superpoly of the cube is 0 is evaluated by numeric mapping. If the superpoly is evaluated as 0, it means that the superpoly is reduced to 0 by annihilating these coefficients. Then, these coefficients are a basis. In the following paper, we call this test a basis test based on the numeric mapping.

For the 844-round Trivium, we set N_0 to 200 and select 2000 cubes of size 38 that contain no adjacent indexes, none of which pass the basis test based on the numeric mapping. The imprecision of numeric mapping limits the power of basis tests. We consider introducing the three-subset division property without unknown subset into the basis test so that the simplified superpolies can be accurately recovered.

Construction of the Division Property Model. Because some expressions are annihilated in the basis test, and the division trails cannot be canceled in the propagation, we cannot initially assign the division property of the key and IV. Then, we consider assigning the division property in the N_0 -round to avoid the cancellation problem. Let $f_1, \ldots, f_t$ be the ANFs corresponding to the state bits in the N_0-round of the cryptographic algorithm. The process of assigning the three-subset division property without unknown subset to the state bits of the N_0-round is depicted in Algorithm 1. For clarity, $\mathrm{T}(f)$ is used to represent all terms in f, and $\mathrm{var}(t)$ is used to represent all variables in term t. We remind the readers that, since $f_1, \ldots, f_t$ are obtained under the condition of a known cube and non-cube IV with a specific assignment, it is unnecessary to consider non-cube IV in Algorithm 1. In the Update function of Algorithm 1, we first copy the required variables and then model "and" and "xor" operations to represent the division property of an ANF. Based on the function initialDP that assigns the division property for ANFs, the basis test based on the division property is given in Algorithm 2.

Determine Whether the Test Passes According to the Solution. In the basis test, for a candidate cube, the MILP model can be established and solved in lines 1–19 of Algorithm 2. Based on the three-subset division property without unknown subset, there are two criteria to determine whether the test passes or fails.

First Criterion: If there are solutions, that is, at least one division trail propagates to the output bit, then the cube fails the test.
Second Criterion: If there are solutions with an odd number of trails, that is, the superpoly has at least one term, then the cube fails the test.

By the above definitions, we can see that the first criterion is more aggressive but also more efficient. The first criterion only requires one feasible solution, so we set the following parameters on Gurobi to focus on finding a feasible solution quickly.

$$\mathcal{M}.\mathrm{MIPFocus} \leftarrow 1$$
$$\mathcal{M}.\mathrm{PoolSearchMode} \leftarrow 1$$
$$\mathcal{M}.\mathrm{PoolSolutions} \leftarrow 1$$

For more on Gurobi and these parameters, readers are requested to refer to [23]. The second criterion requires the enumeration of all possible three-subset trails.

Algorithm 1. The division property of initial key and IV bits

```
procedure UPDATE(MILP model $\mathcal{M}$, $\mathcal{M}.var\ s$, Boolean function $f$)
    Let $count\_k_i(i = 1, \ldots, n)$, $count\_v_j(j = 1, \ldots, m)$, $count\_1$ record the number of variables $k_i$, $v_j$ and constant 1 in boolean function $f$ respectively.
    for $i = 1, \ldots, n$ do                                  ▷ Copy the secret variables
        Declare $\boldsymbol{var\_k_i}$ as $count\_k_i$ MILP variables of $\mathcal{M}$.
        $\mathcal{M}.con \leftarrow k_i = var\_k_{i_1} \vee \cdots \vee var\_k_{i_{count\_k_i}}$
    end for
    for $j = 1, \ldots, m$ do                                  ▷ Copy the public variables
        Declare $\boldsymbol{var\_v_j}$ as $count\_v_j$ MILP variables of $\mathcal{M}$.
        $\mathcal{M}.con \leftarrow v_j = var\_v_{j_1} \vee \cdots \vee var\_v_{j_{count\_v_j}}$
    end for
    Declare $\boldsymbol{var\_1}$ as $count\_1$ MILP variables of $\mathcal{M}$.
    Declare $LinExp \leftarrow 0$ as a MILP linear expression of $\mathcal{M}$.
    $\boldsymbol{loc\_k} \leftarrow \mathbf{0}$, $\boldsymbol{loc\_v} \leftarrow \mathbf{0}$, $loc\_1 \leftarrow 0$    ▷ Record the location of the variables used
    for $t \in \mathrm{T}(f)$ do                    ▷ Add the DP of the terms to the GRBLinExpr
        Declare $temp$ as a MILP variable of $\mathcal{M}$.
        $\mathcal{M}.con \leftarrow temp = var\_k_{i_{loc\_k_i}}$, $loc\_k_i \leftarrow loc\_k_i + 1$  for all $k_i \in \mathrm{var}(t)$
        $\mathcal{M}.con \leftarrow temp = var\_v_{j_{loc\_v_j}}$, $loc\_v_j \leftarrow loc\_v_j + 1$  for all $v_j \in \mathrm{var}(t)$
        $LinExp \leftarrow LinExp + temp$
    end for
    if $1 \in \mathrm{T}(f)$ then              ▷ Add the DP of the constant 1 to the GRBLinExpr
        $LinExp \leftarrow LinExp + var\_1_{loc\_1}$, $loc\_1 \leftarrow loc\_1 + 1$
    end if
    $\mathcal{M}.con \leftarrow s = LinExp$                          ▷ Update the DP of state bits
end procedure
procedure INITIALDP(Cube $I$, Boolean function of $N_0$ round $f_1, \ldots, f_t$)
    Declare an empty MILP model $\mathcal{M}$.
    Declare $\boldsymbol{k}$ as $n$ MILP variables of $\mathcal{M}$ corresponding to secret variables.
    Declare $\boldsymbol{v}$ as $m$ MILP variables of $\mathcal{M}$ corresponding to public variables.
    $\mathcal{M}.con \leftarrow v_i = 1$  for all $i \in I$
    Declare $\boldsymbol{s}$ as $t$ MILP variables of $\mathcal{M}$ corresponding to the state bits of the $N_0$-round.
    for $i = 1, \ldots, t$ do                     ▷ Assign the DP of ANFs to the state bits
        Update($\mathcal{M}$, $s_i$, $f_i$)
    end for
end procedure
```

If there is a high requirement for the basis test, for example, attack the algorithm with a high number of rounds with a small dimension cube, the second criterion should be considered. However, for many candidate cubes, it is time-consuming to test according to the second criterion. Therefore, we use the first criterion for the basis test in this paper.

Computing the Probability. With the basis test based on the division property, we could obtain some cubes with a small number of elements in the basis. For these cubes, we are expected to find f such that the superpoly $p = f \cdot g \oplus h$,

Algorithm 2. The basis test based on the division property

```
procedure basisTEST(Cube $I$, specific assignment to non-cube $IV$, Round $N_0$, Round $R$)
    Set $Q$ to the empty set.
    Initialize the cryptographic algorithm.
    for $i = 1, \dots, N_0$ do
        Compute the ANF of $s_i$ and set $d_i \leftarrow \deg(s_i, I)$.
        Set $Q_i$ to the set of the coefficients of all the terms with degree $d_i$ in the ANF of $s_i$.
        if $d_i \geq 1$ and $1 \notin Q_i$ then
            Set $g = 0$ for each $g \in Q_i$        ▷ Annihilate them to simplify the output
            $Q \leftarrow Q \cup Q_i$
        end if
    end for
    Record the ANF of the state bits of the $N_0$-round as $f_1, \dots, f_t$.
    Declare an empty MILP model $\mathcal{M}$.
    initialDP($\mathcal{M}$,$I$,$f_1, \dots, f_t$)    ▷ Assign the DP to the state bits of the $N_0$-round
    for $i = N_0 + 1, \dots, R$ do
        Update $\mathcal{M}$ according to round functions.
    end for
    Update $\mathcal{M}$ according to output functions.
    solve MILP model $\mathcal{M}$.
    if pass-test then                        ▷ Determine according to the solution
        return $Q$
    end if
end procedure
```

where h has a high bias. Due to the non-uniqueness of the basis, there may be other bases with few elements for the searched cube. Thus, instead of only computing correlations for superpoly p and the elements in the basis when calculating correlations, we test all low-degree polynomials that occur at low rounds.

5 Applications

In this section, we first briefly describe the stream cipher Trivium, then apply the new method to attack Trivium.

5.1 Description of Trivium

Trivium [24] is a bit-oriented synchronous stream cipher that was one of the eSTREAM hardware-oriented finalists. It contains three nonlinear feedback shift registers with lengths of 93, 84, and 111, so it contains 288 internal states recorded as $\boldsymbol{s}$. When the Trivium algorithm is initialized, the 80-bit key variables

$\boldsymbol{k}$ and the 80-bit public variables $\boldsymbol{v}$ are assigned to the internal state respectively, and the assigning method is as follows.

$$\begin{aligned}(s_1, s_2, \ldots, s_{93}) &\leftarrow (k_1, k_2, \ldots, k_{80}, 0, \ldots, 0),\\ (s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (v_1, v_2, \ldots, v_{80}, 0, \ldots, 0),\\ (s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (0, \ldots, 0, 1, 1, 1).\end{aligned}$$

The pseudo code of the update function is given as follows.

```
$t_1 \leftarrow s_{66} \oplus s_{93} \oplus s_{91}s_{92} \oplus s_{171}$
$t_2 \leftarrow s_{162} \oplus s_{177} \oplus s_{175}s_{176} \oplus s_{264}$
$t_3 \leftarrow s_{243} \oplus s_{288} \oplus s_{286}s_{287} \oplus s_{69}$
$(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
```

After updating the internal state iteratively for 1152 rounds, Trivium starts to output keystream bits, i.e.,

$$z \leftarrow s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288}.$$

5.2 The Attack on 844-Round Trivium

The previous highest round of Trivium with balanced superpoly is 843, and the attacks on Trivium with 844 or more rounds are vague. Therefore, we choose to attack 844-round Trivium.

Searching for Cubes. In [18], Liu et al. found some cubes of sizes 36 and 37, which can be used to attack 835-round Trivium. These cubes perform better than randomly generated ones. Therefore, we add a missing index to 13 cubes of size 37 in [18], and we get 553 different 38-dimensional cubes. Through the basis test of 13 existing 37-dimensional cubes and 533 generated 38-dimensional cubes, we obtain about 40 cubes whose superpolies, after 844 rounds, have a basis containing at most 12 elements. Then, we modify them by randomly shifting and changing some indexes to obtain new candidate cubes. After computations within several days on a desktop computer, we find more than 50 cubes whose superpolies, after 844 rounds, have a basis containing at most 12 elements.

Because we need to guess the value of f according to whether the superpoly is constant under different IVs, we need free non-cube IV bits for these cubes. We use the method in [18] to determine some free IV bits. Then, we discard cubes with less than three free IV bits and retain 24 38-dimensional and seven 37-dimensional cubes.

Computing the Probability. In [3], an interesting observation was given that expressions of the form $k_{i+25}k_{i+26} \oplus k_i \oplus k_{i+27} (0 \leq i \leq 52)$ occur frequently in Trivium with low rounds. Therefore, for the searched cube, we experimentally calculate the conditional probability of the superpoly p and elements in the set Ω, where

$$\Omega = \{k_i, 0 \leq i \leq 79\} \cup \{k_{i+25}k_{i+26} \oplus k_i \oplus k_{i+27}, 0 \leq i \leq 52\} \\ \cup \{k_{i-44}k_{i-43} \oplus k_i \oplus k_{i-42}, 44 \leq i \leq 79\}.$$

In each test, we compute the values of the superpoly p_I for 256 random keys and at most 16 non-cube IVs for each key, and evaluate the conditional probability $\Pr(f = 0|p_I(key, \cdot) \equiv 0)$ and $\Pr(f = 1|p_I(key, \cdot) \not\equiv 0)$ for a random fixed key, where g is a function in the basis of p_I and $p_I(key, \cdot)$ denotes the superpoly p_I restricted at a fixed key. We take all possible values of the free non-cube IV bites, and otherwise we set it to 0. Once a non-zero value of p_I is detected in the experiment, we can skip the remaining non-cube IVs and test the following key. Therefore, our experiment requires a total of $5.19 \times 256 \times (24 \times 2^{38} + 7 \times 2^{37}) \approx 2^{53.16}$ cipher operations, where 5.19 is the average number of IVs needed to be computed for each key.

In Table 3 in Appendix, we list the useful cubes found by the experiment and offer the correlation probability. In particular, three cubes are excluded due to their little contribution to attacks. In Table 3, we use $\Pr(p_I \equiv 0)$ to represent the probability that the superpoly p_I is a zero constant restricted at a fixed key. In column $p_I \equiv 0$ (resp., $p_I \not\equiv 0$), we enumerate the polynomials that are 0 with high probability when the superpoly p_I is a zero constant (resp., not a zero constant), and the threshold of probability is 2/3 (resp., 0.65).

Recovering the Key in the Online Phase. In the online phase, we select at most 16 free non-cube IVs to test whether the superpoly under a fixed key is zero constant. We collect the equations with high probability in set G. Then, the expected time complexity of this phase is less than

$$5.19 \times (23 \times 2^{38} + 5 \times 2^{37}) + p^{-r}2^{80-r} \approx 2^{45.05} + 2^{80-\frac{2}{5}r},$$

where r is the number of equations we choose from G to solve.

To test the effect of the attack, we generate 128 new random keys and perform the attack. Due to the limitation of the correlation property obtained by the experimental test, we appropriately reduce the number of selection equations to improve the attack success rate. Considering that $p^{-r} < \binom{|G|}{r}$ needs to be satisfied, we give the strategy choice of r,

$$r = \begin{cases} r' - 10, & \text{if } r' \geqslant 20; \\ r' - 5, & \text{if } r' \geqslant 10; \\ r' - 2, & \text{if } r' \geqslant 4; \\ 1, & \text{if } 3 \geqslant r' \geqslant 1; \end{cases}$$

where r' is the maximum r' such that $p^{-r'} < \binom{|G|}{r'}$. In our experiments for 128 new random keys, the average value of r is 10.14. Then, about $\frac{2}{5}r = 4.06$ key bits could be recovered on average, and the complexity of recovering the whole key could be reduced to about 2^{76}. Specifically, we can recover ten equations on key bits by 74 trials on average. The attack is valid for more than 70% out of 128 random keys in our experiments. As shown in Table 2, the success probability of the attack can be increased by using smaller systems of equations or larger probability thresholds, at the cost of more attack time.

Table 2. Success probability of the attack for 844-round Trivium

#key bits	4.06	3.88	3.61	3.18	2.83
Success rate	71.1%	77.3%	89.1%	93.0%	97.7%

Next, we give an example of the attack procedure. In the example, the time complexity is better than the average case.

Example 1. Given that the 80-bit secret key is EB AA 8F B4 6C 8D 68 F3 28 13 in hexadecimal, where the most significant bit is k_0 and the least significant bit is k_{79}. For each of the 28 cubes in Table 3, we generate at most 16 different non-cube IVs according to their free IV bits, then calculate whether the superpoly is zero constant. We find that there are 6 cubes having zero superpolies,

$$3, 9, 13, 17, 21, 27.$$

Therefore, we obtain 22 equations,

$$G = \{f_3, f_4 \oplus 1, f_{14} \oplus 1, f_{19} \oplus 1, f_{58}, f_{63} \oplus 1, f_{76}, g_0 \oplus 1, g_3, g_5,\\ g_6, g_7, g_{15}, g_{20}, g_{21}, g_{22}, g_{23} \oplus 1, g_{24}, g_{33}, g_{38}, g_{45} \oplus 1, g_{46}\}.$$

The equations become linear equations that are linearly independent of each other after guessing the values of the following bits,

$$k_{25}, k_{28}, k_{31}, k_{32}, k_{40}, k_{46}, k_{47}, k_{49}, k_{59}, k_{64}, k_{71}.$$

According to the strategy, r is chosen to be 13, that is, we randomly choose 13 equations to solve and enumerate the remaining keys. We repeat this step until the correct key is found. In theory, the expected number of trials for finding the correct solution is less than $p^{-r} \approx 270$. As a matter of fact, 17 equations are true for the secret key, but $f_{19} \oplus 1, f_{58}, g_0 \oplus 1, g_{21}$ and $g_{23} \oplus 1$ are not true. On average, we can obtain the correct key in $\binom{22}{13}/\binom{17}{13} = 209$ attempts. Therefore we can recover the key with time complexity of $2^{45.05} + 209 \times 2^{67} \approx 2^{74.71}$.

6 Conclusion

In this paper, we propose a new correlation cube attack. The new view on building correlations between a superpoly and a low degree polynomial on key variables significantly enhance the power of correlation cube attacks. As an illustration, we apply it to the stream cipher Trivium and give the best key recovery attacks for 844-round Trivium. As the number of initialization rounds of Trivium increases, the superpolies in cube attacks becomes more and more massive. A clear advantage of correlation cube attacks is the avoidance of solving complex superpolies. Hence, proposing a new cube attack variant based on correlation properties is worth working on in the future.

Appendix: The Cubes, Equations and Probabilities

Table 3. The useful cubes in the attack on 844-round Trivium

No.	cube indexes	#cube	Free IV bits	$p_I \equiv 0$	$p_I \not\equiv 0$	$\Pr(p_I \equiv 0)$
1	0, 2, 4, 6, 8, 9, 11, 13, 15, 17, 19, 21, 23, 26, 28, 30, 32, 34, 36, 38, 41, 43, 45, 47, 49, 51, 53, 56, 58, 60, 62, 64, 66, 68, 71, 73, 75, 79	38	1,10,76,77	$f_{10} \oplus 1, f_{21}, f_{22} \oplus 1, f_{58}, f_{60}, g_2, g_4, g_6, g_{19}, g_{21}, g_{26}, g_{32}, g_{34}, g_{47}, h_{58} \oplus 1, h_{79} \oplus 1$	-	0.145
2	0, 1, 2, 4, 6, 8, 11, 13, 15, 17, 19, 21, 23, 26, 28, 30, 32, 34, 36, 38, 41, 43, 45, 47, 49, 51, 53, 56, 58, 60, 62, 64, 66, 68, 71, 73, 75, 79	38	9,10,76,77	$f_{10} \oplus 1, f_{21}, f_{22} \oplus 1, f_{58}, f_{60}, g_4, g_6, g_{12} \oplus 1, g_{19}, g_{21}, g_{23}, g_{32}, g_{34}, h_{58} \oplus 1, h_{79} \oplus 1$	-	0.145
3	1, 3, 5, 7, 8, 10, 12, 14, 16, 18, 20, 22, 25, 27, 29, 31, 33, 35, 37, 40, 42, 44, 46, 48, 50, 52, 55, 57, 59, 61, 63, 65, 67, 70, 72, 74, 76, 78	38	0,2,9,11	g_5, g_{20}, g_{22}	-	0.355
4	0, 2, 4, 6, 9, 11, 13, 15, 17, 19, 21, 24, 26, 28, 30, 32, 34, 36, 39, 41, 43, 45, 47, 49, 51, 54, 56, 58, 60, 62, 64, 66, 69, 71, 73, 75, 77, 79	38	1,7,8,10	g_4, g_6, g_{15}, g_{21}	$g23 \oplus 1$	0.387
5	0, 2, 4, 6, 8, 10, 12, 15, 17, 19, 21, 23, 25, 27, 30, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 75, 77, 79	38	3,5,7	$f_9, f_{10} \oplus 1, f_{11} \oplus 1, f_{12}, f_{31}, f_{53}, f_{54} \oplus 1, f_{57}, f_{59}, f_{65}, f_{76}, f_{77} \oplus 1, g_{15}, g_{20}, g_{21}, g_{23}, g_{38}, g_{40}, g_{45} \oplus 1, g_{48} \oplus 1, h_{66}$	-	0.152
6	0, 1, 2, 4, 7, 9, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	38	6,8,10,77	$f_9, f_{58}, g_2, g_4, g_6, g_{13}, g_{15}, g_{17}, g_{19}, g_{21}, g_{26}, g_{34}, g_{47}, h_{58} \oplus 1$	-	0.176
7	0, 2, 4, 6, 9, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 77, 79	38	7,8,10,20	f_{58}, g_4, g_6, g_{15}	-	0.367
8	0, 1, 2, 4, 6, 9, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	38	8,10,20,77	$f_{21}, f_{58}, g_4, g_6, g_{15}, g_{17}, g_{26}, g_{34}, h_{49}, h_{58} \oplus 1$	-	0.184
9	1, 3, 5, 7, 9, 10, 12, 14, 16, 18, 20, 22, 25, 27, 29, 31, 33, 35, 37, 40, 42, 44, 46, 48, 50, 52, 55, 57, 59, 61, 63, 65, 67, 70, 72, 74, 76, 78	38	2,8,11	-	f_9	0.512
10	0, 2, 4, 7, 9, 10, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	38	1,5,6,8	$f_9, f_{43}, f_{58}, g_2, g_4, g_6, g_{15}, g_{19}, g_{21}, g_{26}, g_{34}, h_{58} \oplus 1$	-	0.191
11	0, 1, 3, 5, 7, 9, 11, 13, 16, 18, 20, 22, 24, 26, 28, 31, 33, 35, 37, 39, 41, 43, 46, 48, 50, 52, 54, 56, 58, 61, 63, 65, 67, 69, 71, 73, 76, 78	38	2,6,79	$f_{29} \oplus 1, f_{42}, f_{56}, f_{58}, f_{60}, f_{61}, f_{67}, g_7, g_{12} \oplus 1, g_{21}, g_{22}, g_{45}, h_{53}, h_{68}$	-	0.133
12	0, 2, 4, 6, 7, 9, 11, 13, 15, 17, 19, 21, 24, 26, 28, 30, 32, 34, 36, 39, 41, 43, 45, 47, 49, 51, 54, 56, 58, 60, 62, 64, 66, 69, 71, 73, 75, 79	38	1,5,8,10	$g_2, g_4, g_6, g_{15}, g_{21}, g_{26}, g_{47}, g_{51}, h_{58} \oplus 1$	-	0.227
13	1, 3, 5, 6, 7, 10, 12, 14, 16, 18, 20, 22, 25, 27, 29, 31, 33, 35, 37, 40, 42, 44, 46, 48, 50, 52, 55, 57, 59, 61, 63, 65, 67, 70, 72, 74, 76, 78	38	0,8,9,11	$f_4 \oplus 1, f_{14} \oplus 1, f_{19} \oplus 1, f_{76}, g_5, g_7, g_{20}, g_{21}, g_{33}, g_{46}$	-	0.207

(*continued*)

Table 3. (*continued*)

No.	cube indexes	#cube	Free IV bits	$p_I \equiv 0$	$p_I \not\equiv 0$	$\Pr(p_I \equiv 0)$
14	0, 1, 2, 4, 6, 9, 11, 13, 15, 17, 19, 21, 24, 26, 28, 30, 32, 34, 36, 39, 41, 43, 45, 47, 49, 51, 54, 56, 58, 60, 62, 64, 66, 69, 71, 73, 75, 79	38	8,10,76,77	$g_4, g_6, g_{15}, g_{21}, g_{26}, g_{51}, h58 \oplus 1$	-	0.301
15	0, 2, 4, 6, 8, 10, 11, 13, 15, 17, 19, 21, 23, 26, 28, 30, 32, 34, 36, 38, 41, 43, 45, 47, 49, 51, 53, 56, 58, 60, 62, 64, 66, 68, 71, 73, 75, 79	38	1,9,76,77	$f_{21}, g_6, g_{19}, g_{21}, g_{32}, g_{47}, h_{58} \oplus 1$	-	0.270
16	0, 2, 4, 7, 8, 9, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	38	6,10,77	g_4, g_6, g_{15}	$g_{45} \oplus 1$	0.422
17	0, 2, 4, 6, 8, 9, 11, 13, 15, 17, 19, 21, 24, 26, 28, 30, 32, 34, 36, 39, 41, 43, 45, 47, 49, 51, 54, 56, 58, 60, 62, 64, 66, 69, 71, 73, 75, 79	38	1,7,10,76	g_6, g_{21}	-	0.414
18	0, 2, 4, 6, 9, 10, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	38	1,7,8,20	$f_{21}, f_{58}, f_{61}, g_4, g_6, g_{13}, g_{15}, g_{34}, h_{49}$	-	0.227
19	0, 2, 4, 7, 9, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 77, 79	38	1,5,6,8	$f_{58}, f_{61}, g_2, g_4, g_6, g_{13}, g_{15}, g_{17}, g_{20}, g_{21}$	-	0.246
20	0, 2, 4, 8, 9, 10, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	38	5,6,7,77	$f_{58}, g_6, g_{15}, g_{21}, g_{26}, h_{79} \oplus 1$	$g_{45} \oplus 1$	0.250
21	0, 2, 4, 6, 7, 9, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	38	1,9,10,76	$f_{58}, g_6, g_{15}, g_{21}$	$g_4 \oplus 1 g_{45} \oplus 1$	0.473
22	0, 2, 4, 6, 9, 11, 13, 15, 17, 19, 21, 24, 26, 28, 30, 32, 34, 36, 39, 41, 43, 45, 47, 49, 51, 54, 56, 58, 60, 62, 64, 66, 69, 71, 73, 75, 76, 79	38	1,9,10,76	$g_6, g_{15}, g_{21}, g_{34}$	$g_{45} \oplus 1$	0.367
23	0, 2, 4, 6, 8, 11, 13, 15, 17, 19, 21, 23, 26, 28, 30, 32, 34, 36, 38, 41, 43, 45, 47, 49, 51, 53, 56, 58, 60, 62, 64, 66, 68, 71, 73, 75, 77, 79	38	1,9,10,76	$g_4, g_6, g_{19}, g_{21}, g_{32}, g_{34}, h_{58} \oplus 1$	-	0.250
24	0, 2, 4, 6, 8, 11, 13, 15, 17, 19, 21, 23, 26, 28, 30, 32, 34, 36, 38, 41, 43, 45, 47, 49, 51, 53, 56, 58, 60, 62, 64, 66, 68, 71, 73, 75, 79	37	1,7,8,10	$f_{10} \oplus 1, f_{16} \oplus 1, f_{21}, f_{58}, f_{60}, g_2, g_4, g_6, g_{19}, g_{21}, g_{26}, g_{34}, h_{58} \oplus 1, h_{79} \oplus 1$	-	0.137
25	0, 2, 4, 6, 9, 11, 13, 15, 17, 19, 21, 24, 26, 28, 30, 32, 34, 36, 39, 41, 43, 45, 47, 49, 51, 54, 56, 58, 60, 62, 64, 66, 69, 71, 73, 75, 79	37	1,7,8,10	$f_{21}, g_4, g_6, g_{15}, g_{21}, g_{26}, g_{51}, h_{58} \oplus 1$	-	0.262
26	0, 2, 4, 6, 9, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	37	1,7,8,10	$f_{55}, f_{58}, g_4, g_6, g_{15}, g_{21}, g_{26}, h_{49}$	-	0.191
27	1, 3, 5, 7, 10, 12, 14, 16, 18, 20, 22, 25, 27, 29, 31, 33, 35, 37, 40, 42, 44, 46, 48, 50, 52, 55, 57, 59, 61, 63, 65, 67, 70, 72, 74, 76, 78	37	0,2,6,8	$f_3, f_{14} \oplus 1, f_{19} \oplus 1, f_{63} \oplus 1, f_{76}, g_0 \oplus 1, g_3, g_5, g_7, g_{20}, g_{21}, g_{22}, g_{24}, g_{33}, g_{38}, g_{46}$	-	0.176
28	0, 2, 4, 7, 9, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 79	37	5,6,8,10	$f_9, f_{58}, g_2, g_4, g_6, g_{13}, g_{15}, g_{17}, g_{19}, g_{21}, g_{26}, g_{34}, h_{47} \oplus 1$	-	0.215

$f_i = k_i, 0 \le i \le 79$

$g_i = k_{i+25} \cdot k_{i+26} \oplus k_i \oplus k_{i+27}, 0 \le i \le 52$

$h_i = k_{i-44} \cdot k_{i-43} \oplus k_i \oplus k_{i-42}, 44 \le i \le 79.$

References

1. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_16
2. Fouque, P.-A., Vannet, T.: Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 502–517. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_26
3. Ye, C., Tian, T.: A new framework for finding nonlinear superpolies in cube attacks against trivium-like ciphers. In: Susilo, W., Yang, G. (eds.) ACISP 2018. LNCS, vol. 10946, pp. 172–187. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93638-3_11
4. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 250–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_9
5. Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 275–305. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_10
6. Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: MILP-aided method of searching division property using three subsets and applications. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 398–427. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_14
7. Ye, C., Tian, T.: Revisit division property based cube attacks: key-recovery or distinguishing attacks? IACR Trans. Symmetric Cryptol. **2019**(3), 81–102 (2019)
8. Ye, C.-D., Tian, T.: Algebraic method to recover superpolies in cube attacks. IET Inf. Secur. **14**(4), 430–441 (2020)
9. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 466–495. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_17
10. Hu, K., Sun, S., Wang, M., Wang, Q.: An algebraic formulation of the division property: revisiting degree evaluations, cube attacks, and key-independent sums. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 446–476. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_15
11. Hu, K., Sun, S., Todo, Y., Wang, M., Wang, Q.: Massive superpoly recovery with nested monomial predictions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13090, pp. 392–421. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92062-3_14
12. He, J., Hu, K., Preneel, B., Wang, M.: Stretching cube attacks: improved methods to recover massive superpolies. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. LNCS, vol. 13794, pp. 537–566. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22972-5_19
13. Ye, C.-D., Tian, T.: A practical key-recovery attack on 805-round trivium. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13090, pp. 187–213. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92062-3_7
14. Sun, Y.: Automatic search of cubes for attacking stream ciphers. IACR Trans. Symmetric Cryptol. **2021**(4), 100–123 (2021)

15. Che, C., Tian, T.: An experimentally verified attack on 820-round trivium. In: Deng, Y., Yung, M. (eds.) Inscrypt 2022. LNCS, vol. 13837, pp. 357–369. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-26553-2_19
16. Dinur, I., Shamir, A.: Breaking grain-128 with dynamic cube attacks. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 167–187. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21702-9_10
17. Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional cube attack on reduced-round Keccak sponge function. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 259–288. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_9
18. Liu, M., Yang, J., Wang, W., Lin, D.: Correlation cube attacks: from weak-key distinguisher to key recovery. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 715–744. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_23
19. Liu, M.: Degree evaluation of NFSR-based cryptosystems. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 227–249. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_8
20. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_12
21. Todo, Y., Morii, M.: Bit-based division property and application to Simon family. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_18
22. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24
23. Gurobi Optimization. http://www.gurobi.com
24. De Cannière, C., Preneel, B.: Trivium. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 244–266. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_18

The Triangle Differential Cryptanalysis

Xiaofeng Xie and Tian Tian(✉)

Information Engineering University,
Zhengzhou 450001, China
tiantian_d@126.com

Abstract. In this paper, a new variant of differential cryptanalysis is developed by applying the idea of the boomerang attack on the truncated differential. We call this variant a triangle differential cryptanalysis since it utilizes the difference of every pair in an input and output triple. Similar to the boomerang attack, the triangle differential cryptanalysis combines two independent truncated differential distinguishers of two parts of a cryptosystem into a distinguisher of the whole cryptosystem. It provides a new perspective on the differential propagation, and so it is possible to break the limit of the traditional truncated differential. An MILP modeling technique is also provided for the triangle differential distinguisher search against general SPN ciphers. To demonstrate the power of this new type of differential distinguishers, we apply it to SKINNY-64 and CRAFT. For SKINNY-64, an 11-round triangle differential distinguisher is obtained, while the previous longest truncated differential distinguisher is 10-round. For CRAFT, a 13-round triangle differential distinguisher is obtained, while the previous longest truncated differential distinguisher is 12-round. Besides, compared with the best distinguishers other than the truncated differential distinguishers, there are still some improvements on the probabilities for both SKINNY-64 and CRAFT.

Keywords: Differential cryptanalysis · Boomerang attack · Truncated differential · SKINNY-64 · CRAFT

1 Introduction

Differential cryptanalysis is a powerful method which is widely used in the analysis against block ciphers since its proposal in 1994 [15]. The main idea of differential attacks is utilizing the propagation of differences through encryption components to construct distinguishers with a high probability. To target different types of block ciphers, variants of differential attacks have been proposed. The differential cryptanalysis that considers word-wise differences with high probabilities is called the truncated differential. In some cases, the truncated differential can offer more efficient distinguishers than the bit-based differential. For example, the differential attacks against KLEIN [11,16]. Differential trails with zero probability, both for bit-wise and word-wise trails, could also be used in differential

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 72–88, 2023.
https://doi.org/10.1007/978-3-031-35486-1_4

cryptanalysis, called the impossible differential. The impossible differential has a remarkable effect on AES-like ciphers since many word-wise differentials are impossible for AES-like ciphers with low round numbers. In recent years, a new kind of differential cryptanalysis called the mixture differential was presented in [12], leading to a 4-round mixture differential distinguisher and the best attack on 5-round AES [4]. Based on the 4-round mixture differential distinguisher of AES, the first 6-round AES distinguisher was given in [5]. To construct a long distinguisher based on some shorter distinguishers, the boomerang attacks are proposed. The boomerang attack is always used to combine two arbitrary differential distinguishers [21]. For example, if there exist n-round and m-round distinguishers with probability p and q, respectively, we can naturally construct an $(n+m)$-round boomerang distinguisher whose probability is p^2q^2. Inspired by the mixture differential, a variant of boomerang attack called the retracing boomerang attack was proposed in [9]. The authors of [9] also illustrated that the mixture differential was a special boomerang attack. According to the results of mixture differential against AES, it can be seen that the boomerang attack is one of the most potential cryptanalysis against AES-like ciphers.

Recently, the distinguisher search based on automatic methods has become a prevalent issue. The Mixed-Integer Linear Programming (MILP) method is one of the most effective tools and is widely used to search distinguishers for variants of cryptanalysis such as differential [19], integral [22], linear [10] and meet-in-the-middle attacks [3]. Among the differential cryptanalysis we discussed above, the MILP-based distinguisher search for the truncated differential has already been solved in [17], leading to a series of distinguishers for SKINNY-64, Midori64, and CRAFT. Note that the probabilities of all the truncated differential distinguishers from [17] are higher than the upper bounds of the bit-wise differential characteristics given by [2,6,7], for the same number of rounds. This shows the importance of the truncated differential. For SKINNY-64, Midori64, and CRAFT, the traditional truncated differential distinguishers given by [17] is challenging to improve.

In this paper, we propose a new variant of differential cryptanalysis that applies the idea of boomerang attacks to the truncated differential. We call this differential cryptanalysis as "Triangle Differential". The triangle differential utilize the relationship between the relative differences of 3-tuple in the middle of a cipher. Once we know two of the relative differences of 3-tuple, we can know the last one. Different from the traditional truncated differential that deduces the differential pattern of middle states in only encryption or decryption direction, the triangle differential deduces two of the relative differences in encryption and decryption directions respectively. As a result, it is possible to construct longer distinguishers than the truncated differential. To illustrate the effect of the triangle differential, we applied it to SKINNY-64 and CRAFT using the MILP modeling technique. By solving the MILP models, we get 11- and 13-round triangle differential distinguishers for SKINNY-64 and CRAFT respectively. To the best of our knowledge, the previous best truncated differential distinguishers for SKINNY-64 and CRAFT are 10- and 12-round respectively. Thus, one more round is extended compared with the traditional truncated differential distinguishers. Considering

Table 1. Comparison of characteristics for SKINNY-64 in the single-tweakey model. CP: chosen plaintexts/ACC: chosen plaintexts and adaptively-chosen ciphertexts

Distinguisher	Round	Probability	Data scenario	Ref
Zero-correlation	10	2^{-60}	CP	[18]
Truncated differential	10	2^{-40}	CP	[17]
Impossible differential	11	2^{-60}	CP	[6]
Triangle differential	11	2^{-56}	ACC	Sect. 3

Table 2. Comparison of characteristics for CRAFT in the single-tweakey model. CP: chosen plaintexts/ACC: chosen plaintexts and adaptively-chosen ciphertexts

Distinguisher	Round	Probability	Data scenario	Ref
Linear hull	11	$2^{-58.8}$	CP	[14]
Truncated differential	12	2^{-36}	CP	[17]
bit-wise differential	13	$2^{-59.4}$	CP	[14]
Triangle differential	13	2^{-56}	ACC	Sect. 3

all previously best distinguishers on the two ciphers, although there is no breakthrough on the number of rounds, there are improvements on the probabilities. A summary of the known and the new distinguishers are presented in Tables 1 and 2, where the probability of impossible differential in Table 1 means the effect of the key filtering using the distinguishing, i.e., the probability that a difference falls into the impossible differential in the random case.

This paper is organized as follows. In Sect. 2, a brief introduction to the boomerang attacks and MILP-based truncated differential is given. Some details of SKINNY-64 and CRAFT are also presented as preliminaries. We propose the triangle differential in Sect. 3. In Sect. 4, we apply our new method to SKINNY-64 and CRAFT. A discussion of the potential improvement of our technique is given in Sect. 5. Finally, we summarize the paper in Sect. 6.

2 Background and Previous Work

The proposal of the triangle differential was inspired by the boomerang attack. It applies the idea of boomerang attack to the truncated differential. Thus, an introduction to boomerang attack and truncated differential is given in this section. Moreover, a brief introduction to SKINNY-64 and CRAFT will also be given in this section.

2.1 Notation

For a block cipher F: $\mathbb{F}_2^{n\times m} \rightarrow \mathbb{F}_2^{n\times m}$, let us denote a differential characteristic by $\delta \xrightarrow{F} \triangle$, and the associated differential probability is defined as

$$\Pr[\delta \xrightarrow{F} \triangle] = \frac{|\{x \in \mathbb{F}_2^{n\times m} | F(x) \oplus F(x \oplus \delta) = \triangle\}|}{2^{n\times m}},$$

where $\delta, \triangle \in \mathbb{F}_2^{n\times m}$. The differential characteristic $\delta \xrightarrow{F} \triangle$ satisfying $\Pr[\delta \xrightarrow{F} \triangle] = 0$ is called an impossible differential [8]. For two sets $A, B \subset \mathbb{F}_2^{n\times m}$, the notation $A \xrightarrow{F} B$ denotes the set

$$\{\delta \rightarrow \triangle \ | \text{there exists } x \in \mathbb{F}_2^{n\times m} \text{ such that } F(x \oplus \delta) \oplus F(x) = \triangle, \delta \in A, \triangle \in B\},$$

which is called a truncated differential trail of F. Let $\alpha, \beta \in \mathbb{F}_2^n$. Denote $\alpha \xrightarrow{F} \beta$ as a kind of special truncated differential trails whose input and output sets are subspaces which is active in the bytes indexed by $\{i|\alpha_i = 1\}$ and $\{j|\beta_j = 1\}$ respectively. Furthermore, denote the probability of $\alpha \xrightarrow{F} \beta$ by $\Pr[\alpha \xrightarrow{F} \beta]$.

Definition 1 (Differential pattern). *Let $\alpha, \beta \in \mathbb{F}_2^{n\times m}$ be two states in the middle of a block cipher, where $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$, $\beta = (\beta_0, \beta_1, \ldots, \beta_{n-1})$. Define the differential pattern of (α, β) as $\upsilon(\alpha, \beta) \in \mathbb{F}_2^n$ such that*

$$\upsilon(\alpha, \beta)_i = \begin{cases} 1, \, \alpha_i \oplus \beta_i \neq 0, \\ 0, \, \alpha_i \oplus \beta_i = 0. \end{cases}$$

Remark 1. For $p_0, p_1 \in \mathbb{F}_2^{n\times m}$ and F: $\mathbb{F}_2^{n\times m} \rightarrow \mathbb{F}_2^{n\times m}$, if $\upsilon(p_0, p_1) = a$, then the probability that $\upsilon(c_0, c_1) = b$ is given by $\Pr[a \xrightarrow{F} b]$, where $c_0 = F(p_0)$, $c_1 = F(p_1)$.

2.2 Boomerang Attack

In 1999, Wagner proposed the boomerang attack, which shows how to combine two independent differential distinguishers into a longer distinguisher [21]. The framework of the boomerang attack is depicted in Fig. 1, which can be described as the following theorem.

Theorem 1. *Let E:$\mathbb{F}_2^{n\times m} \rightarrow \mathbb{F}_2^{n\times m}$ be the encryption function of a block cipher, which can be decomposed into $E = E_1 \circ E_0$. Suppose there are differential distinguishers given by*

$$\Pr[\delta_0 \xrightarrow{E_0} \delta_1] = a \textit{ and } \Pr[\delta_3 \xrightarrow{E_1} \delta_2] = b$$

where $\delta_0, \delta_1, \delta_2, \delta_3 \in \mathbb{F}_2^{n\times m}$. Then for p_0, $p_1 \in \mathbb{F}_2^{n\times m}$ satisfying $p_0 \oplus p_1 = \delta_0$, $c_0 = E(p_0)$, and $c_1 = E(p_1)$,

$$\Pr[E^{-1}(c_2) \oplus E^{-1}(c_3) = \delta_0] = a^2b^2,$$

where $c_2 = c_1 \oplus \delta_2$ and $c_3 = c_0 \oplus \delta_2$.

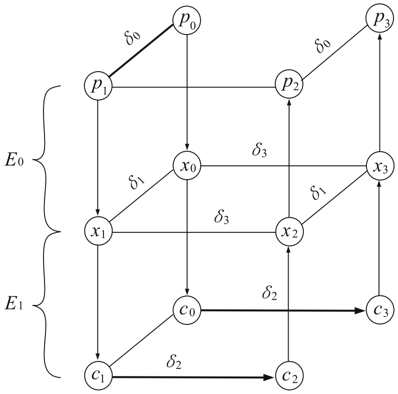


Fig. 1. The framework of the boomerang attack

Proof. For each $i = 0, 1, 2, 3$, let $x_i = E_0(p_i)$. Since $p_0 \oplus p_1 = \delta_0$, $\Pr[x_0 \oplus x_1 = \delta_1] = \Pr[\delta_0 \xrightarrow{E_0} \delta_1] = a$. Since $c_0 \oplus c_3 = c_1 \oplus c_2 = \delta_2$, it follows that

$$\Pr[x_0 \oplus x_3 = \delta_3] = \Pr[x_1 \oplus x_2 = \delta_3] = \Pr[\delta_3 \xrightarrow{E_1} \delta_2] = b.$$

If $x_0 \oplus x_3 = x_1 \oplus x_2 = \delta_3$ and $x_0 \oplus x_1 = \delta_1$, then we have $x_2 \oplus x_3 = x_0 \oplus x_1 = \delta_1$. Assume all these events are independent. Then

$$\Pr[x_2 \oplus x_3 = \delta_1] = \Pr[x_0 \oplus x_3 = \delta_3] \cdot \Pr[x_1 \oplus x_2 = \delta_3] \cdot \Pr[x_0 \oplus x_1 = \delta_1] = ab^2.$$

It follows that

$$\Pr[p_2 \oplus p_3 = \delta_0] = \Pr[x_2 \oplus x_3 = \delta_1] \times \Pr[\delta_0 \xrightarrow{E_0} \delta_1] = a^2b^2.$$

This completes the proof.

It can be seen that in Theorem 1 the difference of the states (x_2, x_3) in the middle is deduced by the following equation

$$x_2 \oplus x_3 = (x_1 \oplus x_0) \oplus (x_1 \oplus x_2) \oplus (x_0 \oplus x_3),$$

where the differences $x_1 \oplus x_0, x_1 \oplus x_2, x_0 \oplus x_3$ are determined by two differential characteristics of E_0 and E_1 respectively, which are assumed to be independent. We note that this provides an idea to connect independent differential characteristics into a longer one. The core idea of the triangle differential proposed in the following paper is also connecting independent differential characteristics but in a different way.

2.3 Truncated Differential Search by MILP

The truncated differential attack was introduced by Knudsen in 1994 [15]. Since it only considers word-wise differences, the propagation of differences about Sbox can be naturally ignored. Thus, it is much more suitable to analyze AES-like ciphers with large Sboxes. Furthermore, since the search space is smaller than the bit-wise situation, it is easier to be modeled by the automatic method. The MILP-based truncated differential cryptanalysis was proposed by Moghaddam et al. in 2019 [19]. An MILP model $\mathcal{M}$ consists of variables $\mathcal{M}.var$, constraints $\mathcal{M}.con$, and an objective function $\mathcal{M}.obj$. The details of constructing the MILP model of the truncated difference can refer to [19]. The key point of the technique is transforming the branching property of a linear layer into linear constraints $\mathcal{M}.con$. Once an MILP model is set up, an MILP solver such as Gurobi could be employed to handle the model. For more details on Gurobi, please refer to [13].

Some results of SKINNY-64 and CRAFT given by [17] are presented in Table 3. The longest round number of truncated differential distinguishers for SKINNY-64 and CRAFT are 10 and 12 rounds. We repeated the experiments given in [17] and observed that since the MILP models covered all the possible distinguishers in the framework of word-wise truncated differentials, only if more details of the encryption components are taken into consideration the differential-style distinguishers could possibly be improved.

Table 3. The longest truncated differential characteristics for SKINNY-64 and CRAFT

Ciphers	Round number	Probability	Ref.
SKINNY-64	10	2^{-40}	[17]
CRAFT	12	2^{-36}	[17]

2.4 SKINNY-64 and CRAFT

The internal states of SKINNY-64 and CRAFT can be viewed as a 4×4 matrix. The round function of the two block ciphers consists of MixColumn(MC), SubCell(SB), Add RoundKey(AK), and Cells Permutation. The nibbles in the states of CRAFT and SKINNY-64 are expressed as

$$S = \begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix}.$$

The round function of CRAFT can be written as $R = SB \circ PN \circ AK \circ MC$, where PN represents the PermuteNibbles operation. The PermuteNibbles permutes the cells of a state as follows:

$$(s_0, \ldots, s_{15}) \longleftarrow (s_{15}, s_{12}, s_{13}, s_{14}, s_{10}, s_9, s_8, s_{11}, s_6, s_5, s_4, s_7, s_1, s_2, s_3, s_0),$$

and the MixColumn transforms every column by multiplying the following matrix

$$M_c = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The matrix M_c satisfies $M_c^{-1} = M_c$. It is worth noting that there is not whitening key in CRAFT.

The round function of SKINNY-64 is $R = MC \circ SC \circ AK \circ SB$ where SC represents the ShiftRow operation. The ShiftRow permutes the cells of a state as follows:

$$(s_0, \ldots, s_{15}) \longleftarrow (s_0, s_1, s_2, s_3, s_7, s_4, s_5, s_6, s_{10}, s_{11}, s_8, s_9, s_{15}, s_{12}, s_{13}, s_{14}).$$

The MixColumn matrix of SKINNY-64 and its inverse matrix are

$$M_s = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \text{ and } M_s^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

3 The Triangle Differential

In this section, we introduce the triangle differential and illustrate its relation to the boomerang attack and the traditional differential. Furthermore, we will discuss its advantage over the traditional truncated differential.

3.1 The Triangle Differential Distinguishers

Let $\alpha, \beta, \gamma \in \mathbb{F}_2^{n \times m}$ be three states in the middle of a block cipher. Suppose we only know $\upsilon(\alpha, \beta)$ and $\upsilon(\beta, \gamma)$. The following lemma gives a way to deduce the differential pattern $\upsilon(\alpha, \gamma)$ according to $\upsilon(\alpha, \beta)$ and $\upsilon(\beta, \gamma)$.

Lemma 1 (Addition of differential pattern). *Let $\alpha, \beta, \gamma \in \mathbb{F}_2^{n \times m}$. Then*

$$\upsilon(\alpha, \gamma)_i = \begin{cases} 0, & \textit{if } \upsilon(\alpha, \beta)_i = \upsilon(\beta, \gamma)_i = 0 \\ 1, & \textit{else if } \upsilon(\alpha, \beta)_i \neq \upsilon(\beta, \gamma)_i, \\ \textit{unknown}, & \textit{otherwise.} \end{cases}$$

Proof. If $\upsilon(\alpha, \beta)_i = \upsilon(\beta, \gamma)_i = 0$, then it follows from $\alpha_i = \beta_i = \gamma_i$ that $\upsilon(\alpha, \gamma)_i = 0$. If $\upsilon(\alpha, \beta)_i = 1$ and $\upsilon(\beta, \gamma)_i = 0$, then

$$\alpha_i \oplus \gamma_i = \alpha_i \oplus \beta_i \oplus \beta_i \oplus \gamma_i = \alpha_i \oplus \beta_i \neq 0,$$

i.e., $\upsilon(\alpha, \gamma)_i = 1$. If $\upsilon(\alpha, \beta)_i = 0$ and $\upsilon(\beta, \gamma)_i = 1$, then the result can be proved similarly.

Remark 2. For a random triple (α, β, γ), the probability that $\upsilon(\alpha, \gamma)_i = 0$ is 2^{-m} in the case that $\upsilon(\alpha, \beta)_i = \upsilon(\beta, \gamma)_i = 1$. Since this probability generally has no influence on the distinguisher search in our applications, we ignore this situation in the following and we simply write $\upsilon(\alpha, \gamma) = \upsilon(\alpha, \beta) \vee \upsilon(\beta, \gamma)$.

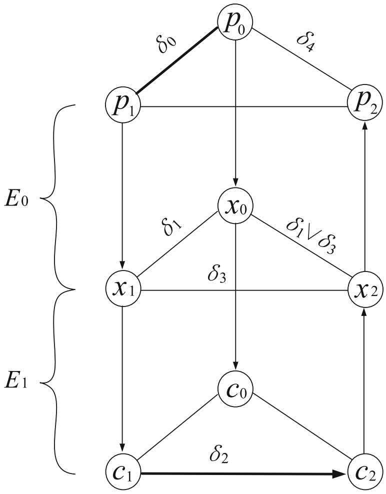


Fig. 2. The framework of the triangle differential

Now we give the framework of the triangle differential in the following theorem, which is also depicted in Fig. 2.

Theorem 2. *Let $E : \mathbb{F}_2^{n\times m} \rightarrow \mathbb{F}_2^{n\times m}$ be an encryption function which can be decomposed into $E = E_1 \circ E_0$. Suppose there are truncated differential characteristics as follows*

$$\begin{aligned} \Pr[\delta_0 \xrightarrow{E_0} \delta_1] \quad &= a, \\ \Pr[\delta_2 \xrightarrow{E_1^{-1}} \delta_3] \quad &= b, \\ \Pr[\delta_1 \vee \delta_3 \xrightarrow{E_0^{-1}} \delta_4] &= c, \end{aligned}$$

where $\delta_0, \delta_1, \delta_2, \delta_3, \delta_4 \in \mathbb{F}_2^n$. Then, for a plaintext pair $(p_0, p_1) \in \mathbb{F}_2^{n\times m} \times \mathbb{F}_2^{n\times m}$ satisfying $\upsilon(p_0, p_1) = \delta_0$ and the corresponding ciphertext pair $c_0 = E(p_0), c_1 = E(p_1)$,

$$\Pr[\upsilon(p_0, p_2) = \delta_4 | \upsilon(c_2, c_1) = \delta_2] = a \times b \times c$$

where $c_2 = E(p_2)$.

Proof. For each $0 \leq i \leq 3$, let $x_i = E_0(p_i)$. Since $\upsilon(p_0, p_1) = \delta_0$, according to $\Pr[\delta_0 \xrightarrow{E_0} \delta_1] = a$, we have

$$\upsilon(x_0, x_1) = \delta_1 \tag{1}$$

with probability a. Similarly,

$$\upsilon(x_1, x_2) = \delta_3 \tag{2}$$

with probability b. If both Eq. (1) and (2) hold, according to Lemma 1 we have

$$\upsilon(x_0, x_2) = \delta_1 \vee \delta_3.$$

Then, according to the truncated differential characteristic $\Pr[\delta_1 \vee \delta_3 \xrightarrow{E_0^{-1}} \delta_4] = c$, we have $v(p_0, p_2) = \delta_4$ with probability c. Hence, assuming that all these events are independent, we have

$$\Pr[v(p_0, p_2) = \delta_4] = a \times b \times c.$$

This completes the proof.

For $\alpha \in \mathbb{F}_2^n$, denote $wt(\alpha)$ as the Hamming weight of α. Thus, for the triangle difference described in Theorem 2, $wt(\delta_4)$ implies the number of active S-boxes in the output difference. The probability that an output pair of a random permutation falls into the distinguisher is $P_{rand} = 2^{-m\times(n-wt(\delta_4))}$. To ensure that the triangle difference can distinguish the block cipher from a pseudo-random permutation, there must be $a \times b \times c > P_{rand}$.

There may be a question why the triangle differential can yield a longer distinguisher than the traditional truncated differential. Let $\delta_0, \delta_1, \delta_2, \delta_3, \delta_4$ be as described in Theorem 2. Suppose $\delta_4 = \delta_0$. Since there is a truncated differential $\delta_0 \xrightarrow{E_0} \delta_1$, the truncated differential $\delta_1 \xrightarrow{E_0^{-1}} \delta_4$ is a possible characteristic with probability c. Moreover, the probability $\Pr[\delta_0 \xrightarrow{E_0} \delta_1] = a$ can be very close to 1, and the number of active Sboxes for $wt(\delta_4) = wt(\delta_0)$ can be very small. On the other hand, since $n - wt(\delta_4)$ is large enough, if there exists a truncated difference $\delta_2 \xrightarrow{E_1^{-1}} \delta_3$ with a large probability b and satisfying $\delta_3 \wedge \delta_1 = \delta_1$, then the constraints $a \times b \times c > 2^{-m\times(n-wt(\delta_4))}$ can easily hold. In another view, a triangle difference combines the following two truncated differences

$$\delta_2 \xrightarrow{E_1^{-1}} \delta_3, \delta_1 \xrightarrow{E_0^{-1}} \delta_0,$$

where $\delta_3 \wedge \delta_1 = \delta_1$. There are many differential patterns for middle states that can not be propagated by any truncated differential trails, i.e., some values for δ_1 and δ_3 are impossible. Triangle differential provides a way to construct differential pattern for middle states which is impossible before. Thus, we can have a larger search space than truncated differential. After the construction of the triangle differential distinguisher with the probability $a \times b \times c$, Algorithm 1 shows the procedure of distinguishing a cipher E from random permutations.

Remark 3. Note that a d-differential distinguisher in the case of $d = 2$ [20] also uses three texts. However, the triangle differential distinguishers are quite different from the 2-differential distinguishers. First, the triangle differential considers relative differences of three texts to each other, while the 2-differential only considers relative differences of two texts with respect to the chosen reference text. Second, the triangle differential considers the propagation of differences for both forward and backward, while the 2-differential only considers one direction (encryption or decryption).

Algorithm 1. The Triangle Differential Attack Algorithm

```
Initialize a counter $ctr \leftarrow 0$.
Generate $(a \times b \times c)^{-1}$ unique pairs $(P_0, P_1)$ with input difference $\delta_0$.
for all pairs $(P_0, P_1)$ do
    Ask for the encryption of $(P_0, P_1)$ to $(C_0, C_1)$.
    Compute $C_2 = C_1 \oplus \delta_2$.
    Ask for the decryption of $C_2$ to $P_2$.
    if $\upsilon(C_0 \oplus C_2) = \delta_4$ then
        Increment $ctr$.
    end if
end for
if $ctr \geq 1$ then
    return This is the cipher $E$.
else
    return This is a random permutation.
end if
```

3.2 MILP Modeling for the Triangle Differential Distinguisher Search

In this section, we are only concerned with the AES-like ciphers. As mentioned before, the point of modeling the distinguisher search of the truncated differential is how to transform the branching property of block ciphers to linear constraints $\mathcal{M}.con$. For a matrix, the branching property is defined as follows.

Definition 2 (Branching property). *Let M be an $m \times m$ matrix on $\mathbb{F}_2^m$, $\alpha, \beta \in \mathbb{F}_2^{n \times m}$. Denote $P_m(a_0, a_1) = \Pr[\upsilon(M \cdot \alpha \oplus M \cdot \beta) = a_1 | \upsilon(\alpha \oplus \beta) = a_0]$. Define the branching property function η of M as follows. For every $(a_0, a_1) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$,*

$$\eta(a_0, a_1) = \begin{cases} 1 - P_m(a_0, a_1), & \text{if } P_m(a_0, a_1) \in \{0, 1\}, \\ -log_2(P_m(a_0, a_1)), & \text{otherwise.} \end{cases}$$

The branching property tables of M_s and M_c can refer to [17]. Beside the encryption, the decryption of a block cipher should also be modeled to the MILP problem. Thus, the branching property of M_s^{-1} and M_c^{-1} are also needed. Since $M_c^{-1} = M_c$, we only present the branching property of M_s^{-1}, see Table 4. The probabilities utilized in branching properties are given by reasonable approximation. Since for M_s, M_c and M_c^{-1}, the probability $P_m(a_0, a_1)$ always close to the values in $\{0, 1, 2^{-m}, 2^{-2m}\}$. We use values in $\{0, 1, 2^{-m}, 2^{-2m}\}$ to estimate $P_m(a_0, a_1)$. This approximation has little impact on the distinguisher search. With the branching property table, we employ Logic Friday [1] to construct a Boolean function $f(x) : \mathbb{F}_2^{10} \rightarrow \mathbb{F}_2$ satisfying

$$f(x) = \begin{cases} 1, \text{ if } \eta(a_0, a_1) = x_8 \times 4 + x_9 \times 8, \\ 0, \text{ otherwise,} \end{cases}$$

where $x = (x_0, x_1, \ldots, x_9), a_0 = (x_0, x_1, x_2, x_3)$ and $a_1 = (x_4, x_5, x_6, x_7)$, thus, $x = a_0 || a_1 || (x_8, x_9)$. Transform f to the product-of-sum representation by Logic

Friday so that we can get the inequalities $l^{(MC)}(x, y, p_0, p_1)$ whose solution corresponds to the branching property, where $x, y \in \mathbb{F}_2^4$, $p_0, p_1 \in \mathbb{F}_2$.

After converting the branching property to linear constraints $l^{(MC)}(x, y, p_0, p_1)$, we can construct an MILP model to search triangle differential distinguishers for $N = N_1 + N_2$ round SKINNY-64 and CRAFT by Algorithm 2, which could be easily generalized to other AES-like ciphers.

Algorithm 2. Triangle differential distinguisher search by MILP

```
for $r = 0$ to $N_1$ do
    $\mathcal{M}.var \leftarrow (x_0^r, x_1^r, \ldots, x_{15}^r)$,
    $\mathcal{M}.var \leftarrow (p_{0,0}^r, p_{1,0}^r, p_{0,1}^r, p_{1,1}^r, p_{0,2}^r, p_{1,2}^r, p_{0,3}^r, p_{1,3}^r)$.
end for
for $r = 0$ to $N_2$ do
    $\mathcal{M}.var \leftarrow (y_0^r, y_1^r, \ldots, y_{15}^r)$,
    $\mathcal{M}.var \leftarrow (p_{2,0}^r, p_{3,0}^r, p_{2,1}^r, p_{3,1}^r, p_{2,2}^r, p_{3,2}^r, p_{2,3}^r, p_{3,3}^r)$.
    $\mathcal{M}.var \leftarrow (z_0^r, z_1^r, \ldots, z_{15}^r)$,
    $\mathcal{M}.var \leftarrow (p_{4,0}^r, p_{5,0}^r, p_{4,1}^r, p_{5,1}^r, p_{4,2}^r, p_{5,2}^r, p_{4,3}^r, p_{5,3}^r)$.
end for
$\mathcal{M}.con \leftarrow \sum_{i=0}^{15} x_i^0 \geq 1$
for $r = 0$ to $N_1 - 1$ do
    for $i = 0$ to $3$ do
        $\mathcal{M}.con \leftarrow l^{(MC)}(x_{Col(i)}^{r+1}, x_{SC(Col(i))}^r, p_{0,i}^r, p_{1,i}^r)$
                    ▷ The notation $Col(i)$ denotes the set $\{j_k|s_{j_k}$ in the $i$-th column $\}$
                        ▷ The notation $SC(Col(i))$ denotes the set $\{SC(j)|j \in Col(i)\}$
        $\mathcal{M}.con \leftarrow l^{(InvMC)}(z_{Col(i)}^r, z_{SC(Col(i))}^{r+1}, p_{4,i}^r, p_{5,i}^r)$
    end for
end for
for $r = 0$ to $N_2 - 1$ do
    for $i = 0$ to $3$ do
        $\mathcal{M}.con \leftarrow l^{(InvMC)}(y_{Col(i)}^r, y_{SC(Col(i))}^{r+1}, p_{2,i}^r, p_{3,i}^r)$
    end for
end for
$\mathcal{M}.con \leftarrow z^0 = x^{N_1} \wedge y^{N_2}$
$P_T = \sum_{r=0}^{n-1} \sum_{i=0}^{4} 4 \times (p_{0,i}^r + p_{2,i}^r + p_{4,i}^r) + 8 \times (p_{1,i}^r + p_{3,i}^r + p_{5,i}^r))$
$P_{rand} = 4 \times \sum_{i=0}^{15} (1 - z_i^{N_1})$
$\mathcal{M}.con \leftarrow P_T < P_{rand}$
$\mathcal{M}.obj \leftarrow Min\{P_T\}$
Solve M;
if M is feasible then
    return $\mathcal{M}.obj$.
end if
```

4 Applications to SKINNY-64 and CRAFT

To illustrate the effect of the triangle differential, we apply this new cryptanalysis on SKINNY-64 and CRAFT in this section. Since the difference of the input and output differences of the linear operation can be deduced from each other, the

Table 4. Branching property table of M_s^{-1}

In/out	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0x2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0x3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0x4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0x5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0x6	0	0	0	0	0	4	0	1	0	0	0	0	0	0	0	0
0x7	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	1
0x8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0x9	0	0	0	0	0	0	4	0	0	0	0	0	0	0	1	0
0xa	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	1
0xb	0	8	0	0	0	0	0	4	0	4	0	0	0	0	0	1
0xc	0	0	0	0	0	0	0	0	0	0	0	0	4	0	1	0
0xd	0	0	0	0	8	0	4	0	0	0	0	0	4	0	1	0
0xe	0	0	0	0	0	0	0	0	0	0	0	4	0	4	0	1
0xf	0	0	0	8	0	8	0	4	0	0	0	4	0	4	0	1

linear operation before the first Sbox or after the last Sbox will be omitted when searching the triangle differential distinguisher. We call these linear operations an independent linear layer.

Let $R = MC \circ SC \circ AK \circ SB$ be the round function of SKINNY-64. The 11-round encryption of SKINNY-64 without an independent linear layer can be written as

$$E = SC \circ SB \circ R^9 \circ MC \circ AK \circ SB.$$

We decompose E into $E = E_1 \circ E_0$, where

$$\begin{aligned} E_0 &= SC \circ AK \circ SB \circ R^4 \circ MC \circ AK \circ SB, \\ E_1 &= SC \circ SB \circ R^4 \circ MC. \end{aligned}$$

Let

$$\begin{aligned} \delta_0 &= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1), \\ \delta_1 &= (1,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1), \\ \delta_2 &= (0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0), \\ \delta_3 &= (1,1,1,0,1,0,0,1,1,1,0,0,0,1,1,1), \\ \delta_4 &= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1). \end{aligned}$$

By employing Gurobi to handle the distinguisher search model, we get the following truncated differential characteristics which can be composed into a triangle differential distinguisher.

$$\begin{aligned} \Pr[\delta_0 \xrightarrow{E_0} \delta_1] &= 1, \\ \Pr[\delta_2 \xrightarrow{E_1^{-1}} \delta_3] &= 2^{-12}, \\ \Pr[\delta_1 \vee \delta_3 \xrightarrow{E_0^{-1}} \delta_4] &= 2^{-44}. \end{aligned}$$

Thus, the probability of distinguisher is 2^{-56}, while $P_{rand} = 2^{-m\times(n-wt(\delta_4))} = 2^{-60}$. More details of 11 round triangle differential distinguisher of SKINNY-64 are presented in Fig. 3.

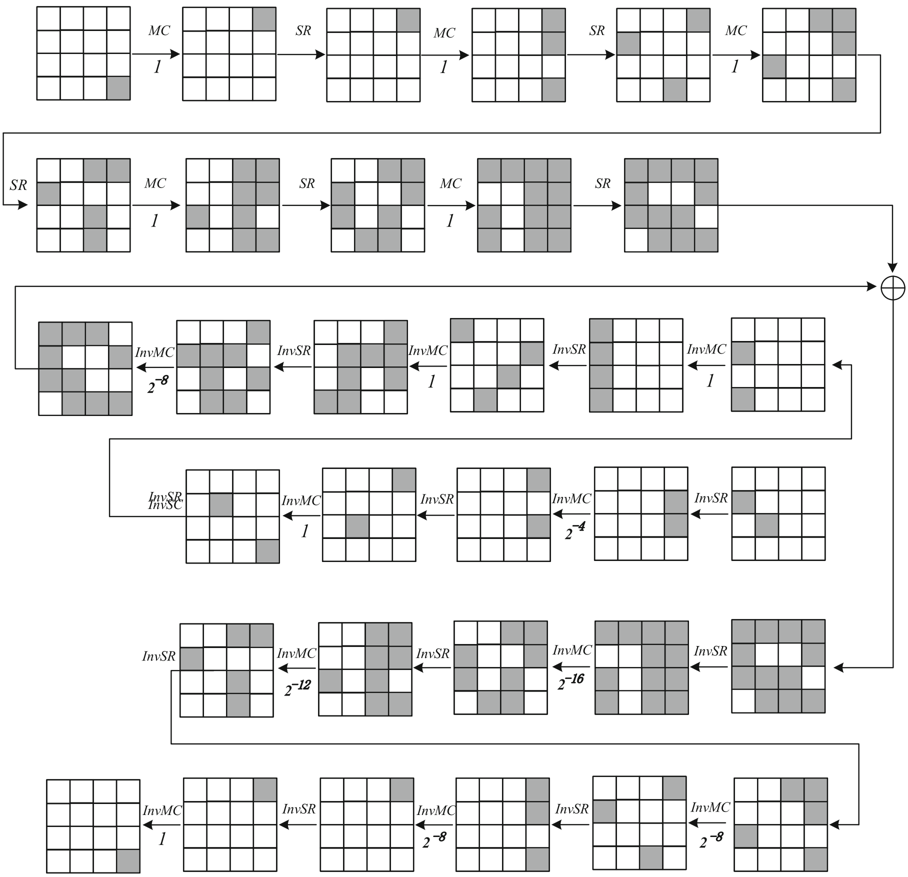


Fig. 3. The triangle difference of SKINNY-64

Denote the 13-round encryption of CRAFT without the first linear layer as $E = R^{12} \circ SB$, where $R = SB \circ PN \circ AK \circ MC$ be the round function of CRAFT. Let $E_0 = R^8 \circ SB$ and $E_1 = R^4$. Thus, $E = E_1 \circ E_0$. Let

$$\begin{aligned}
\delta_0 &= (0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,1),\\
\delta_1 &= (0,0,0,1,0,1,0,0,1,0,0,0,0,1,0,0),\\
\delta_2 &= (0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0),\\
\delta_3 &= (1,1,0,1,0,1,0,0,1,0,0,1,0,1,0,0),\\
\delta_4 &= (0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0).
\end{aligned}$$

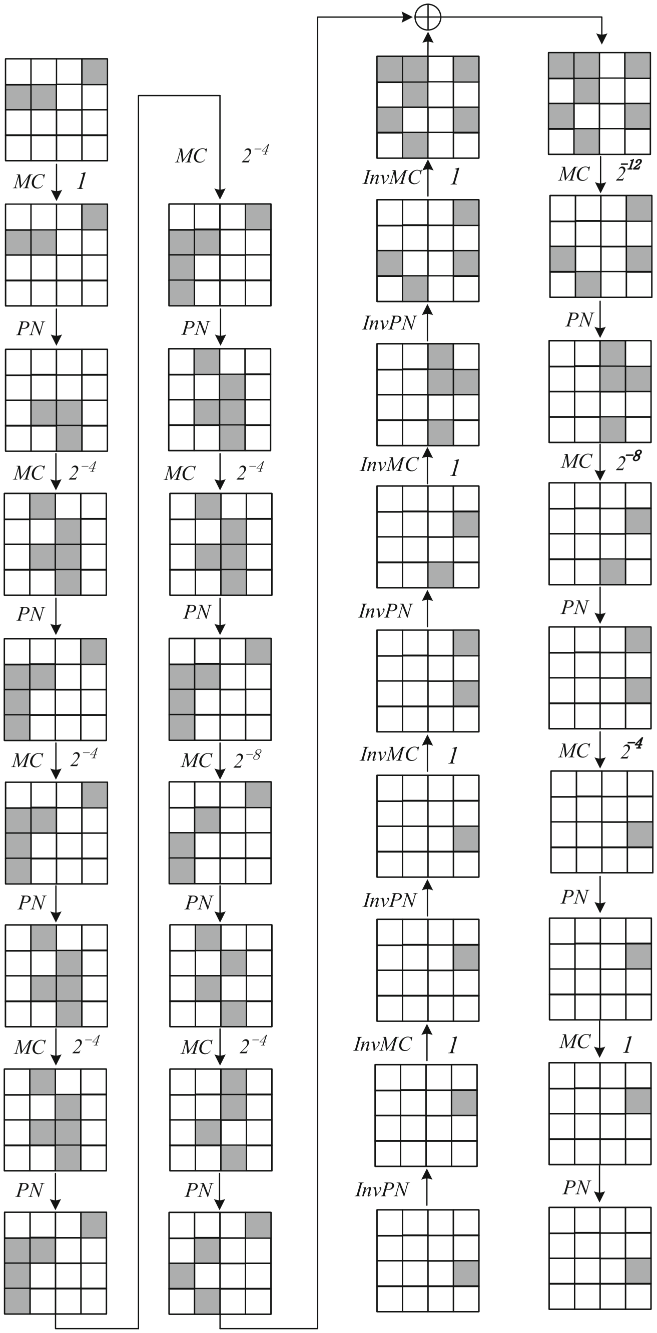


Fig. 4. The triangle difference of CRAFT

The following truncated differential distinguishers for E_0 and E_1 form a triangle differential distinguisher.

$$\begin{aligned} \Pr[\delta_0 \xrightarrow{E_0} \delta_1] &= 2^{-32}, \\ \Pr[\delta_2 \xrightarrow{E_1^{-1}} \delta_3] &= 1, \\ \Pr[\delta_1 \vee \delta_3 \xrightarrow{E_1} \delta_4] &= 2^{-24}. \end{aligned}$$

Since $wt(\delta_4) = 1$, $P_{rand} = 2^{-m\times(n-wt(\delta_4))} = 2^{-60}$. The probability of distniguisher is $2^{-56} > 2^{-60}$. More details of 13 round triangle differential distinguisher of CRAFT are presented in Fig. 4.

It is remarkable that the results of our distinguisher search for SKINNY-64 and CRAFT are both better than the best result of the truncated differential, while the longest truncated differential distinguishers of SKINNY-64 and CRAFT is 10 and 12 round, respectively. We use the Gurobi to solve our MILP based distinguisher search model, and the code of the experiments is presented at https://github.com/BLOCKCIPHERS702702.

5 Further Discussion on the Improvement

In this section, we would like to discuss a potential idea to improve our method, although it did not work in our experiments on SKINNY-64 and CRAFT.

Let E: $\mathbb{F}_2^{n\times m} \to \mathbb{F}_2^{n\times m}$ be the encryption of a block cipher that can be decomposed into $E = E_1 \circ E_0$, and $\delta_0 \xrightarrow{E_0} \delta_1, \delta_2 \xrightarrow{E_1^{-1}} \delta_3$ be truncated differences. For middle states x_0, x_1, x_2 satisfying $v(x_0, x_1) = \delta_1$ and $v(x_1, x_2) = \delta_3$. Denote the i-th bit of δ by $\delta[i]$. Set $\delta_4 = x_0 \oplus x_2$. According to Remark 2 below Lemma 1, we treat $\delta_4[i] = 1$ if $\delta_0[i] = \delta_3[i] = 1$. Practically, $\delta_4[i] = 0$ holds with a small probability in the case of $\delta_0[i] = \delta_3[i] = 1$, and the probability is given by

$$\Pr[\delta_4[i] = 0 \mid \delta_0[i] = \delta_3[i] = 1] = 2^{-m}.$$

It can be seen that if $\delta_4[i] = 0$, then the number of active Sboxes in the middle state is reduced. Thus, although $\delta_4[i] = 0$ holds with a small probability when $\delta_0[i] = \delta_3[i] = 1$, we still think this case might be helpful for searching distinguishers against certain block ciphers since more differential trails are considered by the tradeoff between the probability and the number of active Sboxes.

6 Conclusion

In this paper, we propose a new variant of differential cryptanalysis called triangle differential and illustrate this new cryptanalytic technique against block ciphers. An MILP model is provided for the distinguisher search of triangle differential against the general AES-like cipher. The results of our new technique against SKINNY-64 and CRAFT imply that a triangle differential distinguisher could be longer than a truncated differential distinguisher and also exhibit some advantages over other kinds of distinguishers. The improvement of this technique and applying it to other ciphers will be the subject of future research.

References

1. Logic Friday. https://www.softpedia.com/get/Others/Home-Education/Logic-Friday.shtml
2. Banik, S., et al.: Midori: a block cipher for low energy. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 411–436. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_17
3. Bao, Z., Guo, J., Shi, D., Tu, Y.: Superposition meet-in-the-middle attacks: updates on fundamental security of AES-like hashing. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022. LNCS, vol. 13507, pp. 64–93. (2022). https://doi.org/10.1007/978-3-031-15802-5_3
4. Bar-On, A., Dunkelman, O., Keller, N., Ronen, E., Shamir, A.: Improved key recovery attacks on reduced-round AES with practical data and memory complexities. J. Cryptol. **33**(3), 1003–1043 (2020)
5. Bardeh, N.G., Rønjom, S.: The exchange attack: how to distinguish six rounds of AES with $2^{88.2}$ chosen plaintexts. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 347–370. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_12
6. Beierle, C., et al.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 123–153. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_5
7. 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)
8. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_2
9. Dunkelman, O., Keller, N., Ronen, E., Shamir, A.: The retracing boomerang attack. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 280–309. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_11
10. Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: MILP-based automatic search algorithms for differential and linear trails for speck. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 268–288. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_14
11. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: a new family of lightweight block ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-25286-0_1
12. Grassi, L.: Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. IACR Trans. Symmetric Cryptol. **2018**(2), 133–160 (2018)
13. Gu, Z., Rothberg, E., Bixby, R.: Gurobi optimizer. http://www.gurobi.com/
14. Hadipour, H., Sadeghi, S., Niknam, M.M., Song, L., Bagheri, N.: Comprehensive security analysis of CRAFT. IACR Trans. Symmetric Cryptol. **2019**(4), 290–317 (2019)
15. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60590-8_16

16. Lallemand, V., Naya-Plasencia, M.: Cryptanalysis of KLEIN. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 451–470. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46706-0_23
17. Moghaddam, A.E., Ahmadian, Z.: New automatic search method for truncated-differential characteristics application to Midori, SKINNY and CRAFT. Comput. J. **63**(12), 1813–1825 (2020). https://doi.org/10.1093/comjnl/bxaa004
18. Sadeghi, S., Mohammadi, T., Bagheri, N.: Cryptanalysis of reduced round SKINNY block cipher. IACR Trans. Symmetric Cryptol. **2018**(3), 124–162 (2018)
19. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_9
20. Tiessen, T.: Polytopic cryptanalysis. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 214–239. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_9
21. Wagner, D.: The boomerang attack. In: Knudsen, L. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48519-8_12
22. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24

Key Recovery Attacks on Grain-Like Keystream Generators with Key Injection

Matthew Beighton, Harry Bartlett(✉), Leonie Simpson, and Kenneth Koon-Ho Wong

Queensland University of Technology, Brisbane, QLD, Australia
`matthew.beighton@hdr.qut.edu.au`, `{h.bartlett,lr.simpson}@qut.edu.au`,
`k45.wong@connect.qut.edu.au`

Abstract. A common structure in stream ciphers makes use of linear and nonlinear shift registers with a nonlinear output function drawing from both registers. We refer to these as Grain-like keystream generators. A recent development in lightweight ciphers is a modification of this structure to include a non-volatile key register, which allows key bits to be fed into the state update of the nonlinear register. Sprout and Plantlet are examples of this modified structure. The authors of these ciphers argue that including these key bits in the internal state update provides increased security, enabling the use of reduced register sizes below the commonly accepted rule of thumb that the state size should be at least twice the key size.

In this paper, we analyse Plantlet and show that the security of this design depends entirely on the choice of the output function. Specifically, the contribution from the nonlinear register to the output function determines whether a key recovery attack is possible. We make a minor modification to Plantlet's output function which allows the contents of the linear register to be recovered using an algebraic attack during keystream generation. This information then allows partial recovery of the contents of the nonlinear register, after which the key bits and the remaining register contents can be obtained using a guess and check approach, with a complexity significantly lower than exhaustive key search.

Note that our attack is not successful on the existing version of Plantlet, though it only requires minor modifications to the filter function in order for the attack to succeed. However, our results clearly demonstrate that including the key in the state update during keystream generation does not increase the security of Plantlet. In fact, this feature was exploited to recover the key during keystream generation without the need to consider the initialisation process. This paper provides design guidelines for choosing both suitable output functions and the register stages used for inputs to these functions in order to resist the attacks we applied.

Keywords: Key recovery · algebraic attack · key injection · Plantlet · Grain-like structures · lightweight ciphers

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 89–108, 2023.
https://doi.org/10.1007/978-3-031-35486-1_5

1 Introduction

Symmetric stream ciphers are used to provide confidentiality for a range of real-time applications. The most common type of stream cipher is the binary additive stream cipher, where encryption and decryption are performed by XORing a binary keystream with the plaintext or ciphertext bitstream, respectively. The reciprocal nature of the XOR operation provides high speed encryption and decryption processes. However, the security provided depends on the properties of the keystream. The security of the keystream generator is therefore crucial.

The importance of secure encryption is highlighted by the recent NIST competition for AEAD algorithms suitable for lightweight applications. These algorithms are intended for use in applications such as the ubiquitous Internet of Things (IoT) and require high security levels with reduced computational load and/or component size.

Many keystream generators are based on shift registers with either a linear or nonlinear feedback function, denoted as linear feedback shift registers (LFSRs) and nonlinear feedback shift registers (NFSRs), respectively. However, previous work [3,6,7] has shown that shift register based ciphers are vulnerable to algebraic attacks. In response, some contemporary keystream generators use a combination of a NFSR and a LFSR, together with a nonlinear filter function taking inputs from both registers. We refer to these generators as "Grain-like structures", as the well known Grain family of stream ciphers [13–17] is designed in this way. The combination of LFSRs and NFSRs in these structures was intended to prevent such algebraic attacks; however, Berbain et al. [4] and Beighton et al. [2] have demonstrated viable attacks against Grain-like structures with particular forms of output function.

A further development, aimed at providing encryption for lightweight devices, is to include a non-volatile key register in the cipher state. The key bits stored in this register are then injected into the feedback of the shift registers during operation. The ciphers Sprout [1] and Plantlet [22] both use this technique. This introduction of key injection into Grain-like ciphers was intended to increase the security of these ciphers, so that lightweight ciphers with smaller registers could still provide acceptable security levels. In particular, the internal state size in these ciphers is less than twice the key size, contradictory to the rule of thumb from Hong and Sarkar [18] in relation to generic time-memory trade-off attacks.

The authors of Sprout [1] invited further investigation of these designs featuring key injection. In this paper, we therefore investigate the security provided by Grain-like keystream generators with key injection. Specifically, we explore an algebraic key recovery attack on Plantlet and find that it can be successfully performed with only a minor modification to the output function and with the key injection feature left intact. Our attack is based on the algebraic attack applied to Grain-like structures by Beighton et al. [2]; it is applied during keystream generation, so no knowledge of the initialisation function is required.

This paper is organised as follows: Sect. 2 provides background information on shift register based designs. Section 3 discusses current algebraic attack techniques. Section 4 presents our algebraic attack technique for application to

Grain-like structures with key injection. We then apply our attack technique to modified Plantlet in Sect. 5. Experimental simulations for proof of concept are reported in Sect. 6 and discussed in Sect. 7. Conclusions are drawn in Sect. 8.

2 Preliminaries and Notation

2.1 Feedback Shift Registers

A binary feedback shift register (FSR) of length n is a set of n storage devices called *stages* $(r_0, r_1, ..., r_{n-1})$, each containing one bit, together with a Boolean update function g. The state at any time t is defined to be S_t, where $S_t = s_t, s_{t+1}, ..., s_{t+(n-1)}$, and the sequence of state bits that passes through the register over time is denoted S; that is $S = s_0, s_1, .., s_{n-1}, s_n, ...$.

The shift registers used in the types of keystream generators we discuss in this paper are regularly clocked Fibonacci style, as shown in Fig. 1. Thus, the state update function takes the form:

$$r_i^{t+1} = \begin{cases} r_{i+1}^t & i = 0, 1, \ldots, n-2 \\ g(r_0, r_1, .., r_{n-1}) & i = n-1 \end{cases}$$

If g is linear, the register is said to be a linear feedback shift register (LFSR) and if g is nonlinear, the register is said to be a nonlinear feedback shift register (NFSR).

A binary sequence can be generated from a FSR by applying a Boolean function f to the state S_t, as shown in Fig. 1. Here, the output $y = f(S_t)$ can be a function of the contents of one or more register stages.

2.2 Filter Generators

Keystream generators where f is a function of the contents of multiple stages are called *filter generators*. If f and g are both linear, the filter generator is equivalent to another LFSR, which provides very little security to the plaintext

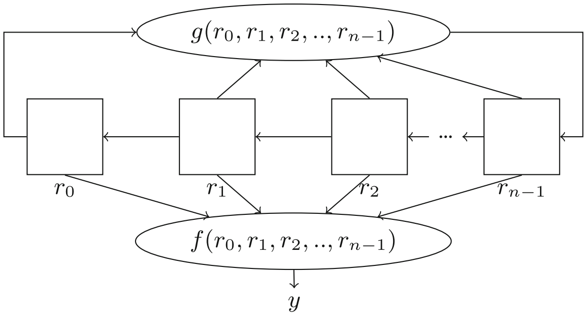

$g(r_0, r_1, r_2, .., r_{n-1})$
...
r_0
r_1
r_2
r_{n-1}
$f(r_0, r_1, r_2, .., r_{n-1})$
y

Fig. 1. An n-stage FSR with update function g and filter function f.

[20]. For this reason, LFSRs were traditionally filtered using a nonlinear Boolean function [19].

A keystream generator consisting of a LFSR and a nonlinear filter function f is known as a *nonlinear filter generator* (NLFG) [24]. These designs have been extensively analysed and are susceptible to numerous attacks, including correlation attacks [10,12,21,24], algebraic attacks [6,7,9] and distinguishing attacks [8]. The underlying LFSR provides only desirable statistical properties for the binary sequence, while the resistance of the NLFG to cryptanalysis is determined by the properties of the nonlinear filter function. As a single nonlinear Boolean function cannot display high levels of all the desirable cryptographic properties [23], choosing a filter function that resists one form of attack may leave the keystream generator vulnerable to other attacks.

In response to the cryptanalysis of NLFGs, designs using NFSRs were proposed. A *linearly filtered nonlinear feedback shift register* (LF-NFSR) [11] has a nonlinear update function g and a linear filter function f, and is the dual construction of the NLFG. Berbain et al. [3] showed that LF-NFSRs are also susceptible to algebraic attacks, resulting in initial state (and possibly secret key) recovery. From Berbain's results, it is clear that the properties of the filter function used in a LF-NFSR are critical in providing resistance to a traditional algebraic attack.

2.3 Composite Combiners and 'Grain-Like' Structures

Effective algebraic attacks have been proposed on both NLFG and LF-NFSR keystream generators. A more complex design incorporates both a LFSR and a NFSR, together with a nonlinear filter function taking inputs from both registers, as shown in Fig. 2. Keystream generators using this structure include Grain [15] and subsequent variants of Grain [13,14,16,17]. We denote this general design as a "Grain-like" structure. Here we consider the lengths of the NFSR and LFSR to be the same (n). However, the approach outlined also applies in the case where register lengths differ.

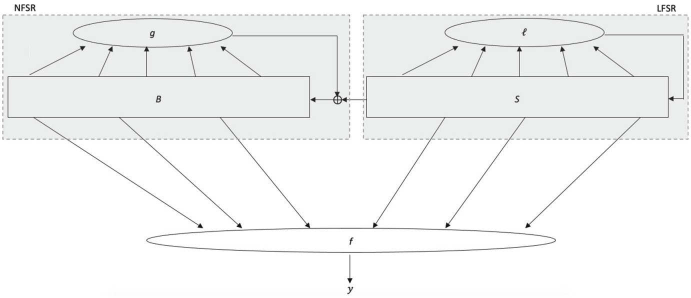


Fig. 2. Grain-like structure.

We denote the states of the NFSR and the LFSR at any time t as B_t and S_t. The sequences of state bits that pass through the registers over time are denoted B and S; that is $B = b_0, b_1, .., b_{n-1}, b_n, ...$ and $S = s_0, s_1, .., s_{n-1}, s_n, ...$ In the case of Grain-like structures, we denote the nonlinear update function as g, the linear update function as ℓ and the filter function as f. For a Grain-like structure, the LFSR is autonomous when producing output as all of the inputs to ℓ are from the LFSR. The NFSR is not autonomous, as the nonlinear update function g contains one input from the LFSR.

The filter function f can be considered as the XOR sum (here denoted '+') of several different types of monomials. That is, we consider sub-functions of f. We define the following sub-functions, each as a sum of the terms indicated:

- L_B - monomials with linear inputs from NFSR.
- L_S - monomials with linear inputs from LFSR.
- f_S - nonlinear monomials with inputs from LFSR only.
- f_B - nonlinear monomials with inputs from NFSR only.
- f_{BS} - nonlinear monomials with inputs from both NFSR and LFSR.

Thus, any filter function f in a Grain-like structure can be expressed as follows:

$$f(B, S) = L_B + L_S + f_B + f_S + f_{BS}. \tag{1}$$

2.4 Grain-Like Structures Using Key Injection

Newer contemporary designs have been proposed which use the general Grain-like structure, but also incorporate the secret key in the state update of the keystream generator, as in Fig. 3. Keystream generators such as those used in Sprout [1] and Plantlet [22] have this design. We denote this general design as a "Grain-like structure with key injection" and use the same notation as for a general Grain-like structure.

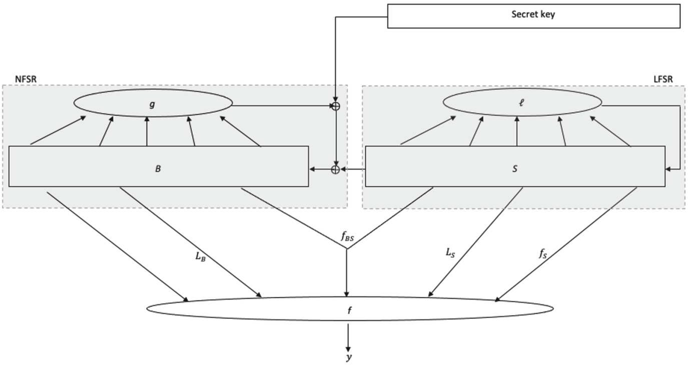


Fig. 3. Grain-like structure with key injection.

3 Current Algebraic Attacks

Algebraic attacks were first introduced by Courtois and Meier [7] on ciphers with linear feedback. The goal of an algebraic attack is to create a system of low degree equations that relates the initial state bits of the cipher to some observed output bits and then to solve these equations to recover the internal state values. For a binary additive stream cipher, the output may be obtained using a known-plaintext attack.

These attacks are performed in two phases: pre-computation and online. The pre-computation phase builds a system of equations relating initial state bits and output bits. In the online phase, given an observed output sequence $\{y_t\}_{t=0}^{\infty}$, the appropriate substitutions are performed, the system is solved and the initial state recovered.

3.1 Algebraic Attacks on NLFGs [7]

Each output bit of a NFSR satisfies the equation

$$y_t = f(S_t) = f(s_t, \ldots, s_{t+n-1}) \tag{2}$$

The linear update function g of the LFSR can then be used to replace state bits s_{t+n-1} with the corresponding linear combination of initial state bits, keeping the equation system of a constant degree $(deg(f))$ while maintaining the number of unknown variables in the system.

In many cases, each equation in the system may be multiplied through by a low degree multivariate function h (of degree e) to reduce the overall degree of the system of equations [7]. If $fh = 0$, then h is defined as an annihilator of f. Each equation in the resulting system has the form

$$f(S_t)h(S_t) = y_t h(S_t).$$

The degree of this system will be equal to $deg(fh) = d$, where $d < deg(f)$, with n independent variables, where n is the length of the underlying LFSR. For a more detailed explanation, the reader is referred to Courtois and Meier's paper [7]. Appendix A.1 provides an algorithm for this attack and also discusses data requirements and computational complexity.

3.2 Fast Algebraic Attacks on NLFGs [6]

Fast algebraic attacks [6] significantly reduce the complexity of the online phase by reducing the overall degree of the system of equations below the degree of fh. This increases the complexity of the precomputation phase; however, precomputation only needs to be performed once for a particular cipher and the equation system may then be reused in multiple online phases to recover the states of the cipher corresponding to multiple different keystreams.

These attacks use a concept Courtois [6] described as "double-decker equations". These equations allow an attacker to equate an expression in terms of initial state bits only to an expression in terms of initial state bits and observed output bits. The technique targets monomials in initial state bits only of degree from $e = deg(h)$ to $d = deg(fh)$: given approximately $\binom{n}{d}$ equations, these monomials will occur in multiple equations and can be replaced by suitable linear combinations of those equations. These linear combinations define a new system in n unknowns, of degree $e < d$. This new system can be solved by linearisation, with less computational complexity than for traditional algebraic attacks.

For a detailed explanation, the reader is referred to Courtois' paper [6]. Appendix A.2 provides an algorithm for this attack and also discusses data requirements and computational complexity.

3.3 Algebraic Attacks on LF-NFSRs

Initially, LF-NFSRs were considered resistant to algebraic attacks, due to the use of a nonlinear state update function. Using the nonlinear feedback function to derive equations for the update bits in terms of initial state bits causes the degree of the system of equations to increase over time. However, Berbain et al. [3] showed that it is possible to keep the degree of the system of equations constant. This allows an algebraic attack which can recover the initial state of the underlying NFSR (and possibly the secret key).

Berbain et al. [3] noted that the equation for the first output bit

$$y_0 = \ell(s_0, ..., s_{n-1}) = \sum_{k=0}^{n-1} a_k s_k,$$

(with $a_k \in \{0, 1\}$) implies that

$$s_j = \sum_{k=0}^{j-1} a_k s_k + y_0.$$

where j is the highest index in the original summation for which $a_j = 1$.

Repeating this process for all subsequent time steps allows us to express every bit, s_{j+t} for $t \geq 0$, as a linear combination of output bits and initial state bits. This produces a set of equations of the form:

$$s_{j+t} = \sum_{k=0}^{j-1} a_{k+t} s_{k+t} + y_t,$$

for $t \geq 0$. Note that if the latter summations contain any term for which an equation already exists, the term can be replaced by the corresponding linear combination of initial state bits and output bits.

Appendix A.3 provides an algorithm for this attack and also discusses data requirements and computational complexity.

3.4 Algebraic Attacks on Grain-Like Structures

After successfully applying algebraic attacks to LF-NFSRs, Berbain et al. [4] proposed an algebraic attack on Grain-like structures where the output function f is the XOR combination of a LF-NFSR and a NLFG. That is, adopting the notation from Sect. 2.3, $f(B,S) = L_B + L_S + f_S$.

In this case the output of the keystream generator can be expressed as

$$y_0 = L_B + L_S + f_S = \sum_{k=0}^{n-1} a_k b_k + L_S + f_S. \qquad (3)$$

As discussed in Sect. 3.3, an equation of the form taken by Eq. 3 can be rearranged as follows:

$$b_j = \sum_{k=0}^{j-1} a_k b_k + L_S + f_S + y_0.$$

Repeating this for $t > 0$ allows for NFSR state bits of index j or higher to be represented as the XOR sum of:

- a linear combination of NFSR initial state bits
- a linear and nonlinear combination of LFSR initial state bits
- a linear combination of observed output bits.

A second system of equations can then be built using the nonlinear update function to the NFSR, making substitutions from the system generated by Eq. 3 where applicable. This system will be of degree at most $deg(g)deg(f_S)$. Combining the two systems results in a system of equations of degree $deg(g)deg(f_S)$ in $n+j$ unknown initial state bits, where n is the size of LFSR and j is the index of the highest indexed term in L_B. The success of this attack in recovering the LFSR and NFSR initial states demonstrated that using the contents of stages in the NFSR linearly in the output function is not sufficient to provide resistance to algebraic attacks; the NFSR contents must also be filtered nonlinearly in some way.

Beighton et al. [2] proposed an algebraic attack on Grain-like structures where the output function f is of the form $f = L_S + f_S + f_{BS}$. They note that, by using the idea of annihilators presented by Courtois and Meier [7], f may be multiplied by a low degree function containing only LFSR bits that will eliminate f_{BS}. This function, denoted A_{BS}, is considered as a "partial annihilator", in as much as A_{BS} only annihilates certain monomials.

Multiplying f by A_{BS} leaves an equation of the form

$$zA_{BS} = A_{BS}(L_S + f_S),$$

which is an equation containing only LFSR initial state bits. Fast algebraic attack techniques can then be applied to recover the LFSR initial state, from which the NFSR initial state can be partially recovered.

4 Our Divide and Conquer Attack on Grain-Like Structures with Key Injection

As highlighted in Beighton et al.'s paper [2], if the filter function of a Grain-like structure does not feature a monomial that takes inputs only from the NFSR and does not divide any other monomial, the structure is vulnerable to a divide and conquer attack which first targets the LFSR in an algebraic attack and then determines the NFSR contents.

This idea extends to the case where the secret key is used to update the NFSR, as the LFSR still runs autonomously.

4.1 Generalised Algebraic Attack Algorithm

We present here the generalised algebraic attack for Grain-like structures with key injection. This attack uses a divide and conquer strategy. We first target the LFSR and recover the LFSR initial state. The NFSR is then targeted, with partial NFSR initial state recovery possible. From this point, we can simultaneously determine the key bits and the remaining NFSR bits.

Recovering the LFSR. Consider a keystream generator that produces an output bit at each time step by:

$$z = L_S + f_S + f_{BS}. \tag{4}$$

That is, NFSR state bits are only used nonlinearly and only in f_{BS}. Every monomial in f_{BS} will contain both NFSR bits and LFSR bits. Thus, using the idea of annihilators presented by Courtois and Meier [7], we may multiply Eq. 4 by a low degree function containing only LFSR bits that will eliminate f_{BS}. We denote this function as A_{BS}, and consider it to be a "partial annihilator" in as much as A_{BS} only annihilates certain monomials. Note that the degree of the NFSR bits in f_{BS} does not affect the ability to annihilate the monomials containing bits from the NFSR.

Therefore Eq. 4 can be rewritten as

$$zA_{BS} = A_{BS}(L_S + f_S), \tag{5}$$

which is an equation containing only LFSR initial state bits. The degree of the system of equations built using Eq. 5 will be at most $deg(A_{BS}) + deg(f_S)$. Note, however, that the right hand side of Eq. 5 contains only initial state bits from the LFSR. This means that fast algebraic attack methods can be performed in the precomputation phase of the attack to reduce the degree of unknown variables in the system from $deg(A_{BS}) + deg(f_S)$ to $deg(A_{BS})$.

There are several other cases where the attack works. For convenience we use only the simplest example here to illustrate the first phase of the attack and refer the reader to Beighton et al.'s paper [2] for a detailed discussion of the other cases.

The structure of a system of equations built in this way allows for the fast algebraic attack techniques highlighted in Sect. 3.2 to be applied. That is, given access to approximately $\binom{n}{d}$ bits of output (where n is the size of the LFSR and d is the algebraic degree of the system relating LFSR initial state bits to observable output bits), a precomputation phase can be performed that allows a new system of equations to be built of degree $e < d$, where e is the degree of A_{BS}. This precomputation phase has a complexity of $\mathcal{O}(\binom{n}{d}log\binom{n}{d}+n\binom{n}{d})$. The initial state of the LFSR can then be recovered in the online phase of the attack by observing approximately $\binom{n}{d}$ bits of output with complexity $\mathcal{O}(\binom{n}{d}\binom{n}{e}+\binom{n}{e}^{\omega})$, where ω is the Guassian elimination exponent $\omega \approx 2.8$.

Recovering the NFSR and the Key. Once the LFSR initial state is recovered, every future LFSR state bit will be known, as the LFSR is autonomous during keystream generation. The next stage is to recover the NFSR initial state. In doing so, we will simultaneously recover the key.

Since the key is used to update the state of the NFSR, recovering the NFSR state at one point in time without knowing the key is not enough to determine the NFSR state at other points in time. As a result, the key must also be taken into account. To begin, however, we will consider the idea of recovering the NFSR state by itself and then expand this approach to recover both the NFSR state and the key.

Consider the example filter function used to produce the keystream bit

$$z = x_1 + x_4x_5 + x_0x_3 + x_0x_1x_2 + x_2x_3x_4x_5, \tag{6}$$

where x_0, x_1, x_2, x_3 are from the LFSR and x_4, x_5 are from the NFSR. Since the LFSR is known, each output bit will have the form

$$z = \alpha x_4x_5 + \beta,$$

where α and β may be 0 or 1, respectively.

Clearly, when $\alpha = 0$ no information about the initial state of the NFSR is leaked. We must therefore utilise the case where $\alpha = 1$. If $z = x_4x_5$ and $z = 1$, then we know $x_4 = x_5 = 1$. Likewise if $z = x_4x_5 + 1$ and $z = 0$, then we know $x_4 = x_5 = 1$. Once we have recovered these state bits, we may then look to equations where $z = x_4x_5$ and $z = 0$, but for which we know either x_4 or x_5 equals 1. We would then know that the unknown state bit is equal to zero. Similarly for the case where $z = x_4x_5 + 1$ and $z = 1$. Continuing in this way, we may be able to recover n consecutive bits of the NFSR.

For certain filter functions it may not be possible to recover n consecutive state bits. In this case, the partially recovered initial state reduces the exhaustive search required to recover the correct initial state of the NFSR. For instance, suppose m bits of the NFSR can be recovered. This leaves 2^{n-m} possible candidates for the correct NFSR initial state which, for $m > 0$, is better than exhaustively searching the entire register. Each candidate can be used (together with the known LFSR initial state) to produce output. The candidate which produces the correct output sequence can be assumed to be the correct initial state.

As stated earlier, because the key is used to update the state of the NFSR, recovering the state of the NFSR at a single point in time is not sufficient to determine the state at any other point in time. However, as we now show, the NFSR state recovery process discussed above may be adapted to recover the NFSR state and the secret key simultaneously by considering a longer sequence of NFSR bits. We consider the sequence of bits that pass through the NFSR as the XOR of some NFSR feedback bits and the key. We thus have,

$$\begin{aligned} b_{t+n} &= g(B_t) + k_{t \bmod |K|} \\ &= b'_{t+n} + k_{t \bmod |K|}. \end{aligned} \tag{7}$$

Thus, given $n+|K|$ consecutive NFSR state bits, the key can easily be recovered.

The goal is therefore to recover $n+|K|$ consecutive bits of the NFSR, or to recover as much as is possible and then exhaustively search the rest. For instance, suppose m bits of the NFSR sequence can be recovered. This leaves $2^{(n+|K|)-m}$ possible candidates for the correct NFSR sequence which, for $m > n$, is better than exhaustively searching the key. Each candidate can be used (together with the known LFSR initial state) to produce further NFSR states and thus output. The candidate which produces the correct output sequence can be assumed to be the correct sequence. From the sequence the key can quickly be recovered.

5 Algebraic Attack on a Modified Version of Plantlet

We now mount an algebraic attack on adapted versions of the stream cipher Plantlet with a modified filter function. We show that even with the key bits used to update the NFSR state, the success of the attack is determined only by the choice of filter function.

5.1 The Plantlet Stream Cipher

Plantlet is a contemporary stream cipher that uses a very small internal state. The keystream generator for Plantlet consists of a LFSR and a NFSR, together with a nonlinear Boolean function that takes inputs from both registers.

Initialisation. Plantlet takes as input a 80-bit secret key and 90-bit IV. In initialisation, the NFSR and the LFSR are 40 bits and 60 bits in length. The keystream generator is loaded by filling the NFSR with 40 IV bits. The remaining 50 bits of the IV are loaded into the LFSR, which is then padded with a constant.

At each time step the LFSR is updated using the linear update function ℓ as follows

$$s_{t+59} = \ell(S_t) = z_t + s_t + s_{t+14} + s_{t+20} + s_{t+34} + s_{t+43} + s_{t+54}. \tag{8}$$

The NFSR is updated at each time step using the nonlinear update function g as follows

$$\begin{aligned}b_{t+39} = g(B_t) = s_t + z_t k_{t \bmod 80} + c_t^4 + b_t + b_{t+13} + b_{t+19} + b_{t+35} + b_{t+39}\\ + b_{t+2}b_{t+25} + b_{t+3}b_{t+5} + b_{t+7}b_{t+8} + b_{t+14}b_{t+21} + b_{t+16}b_{t+18}\\ + b_{t+22}b_{t+24} + b_{t+26}b_{t+32} + b_{t+33}b_{t+36}b_{t+37}b_{t+38}\\ + b_{t+10}b_{t+11}b_{t+12} + b_{t+27}b_{t+30}b_{t+31},\end{aligned} \tag{9}$$

where c_t^4 is the fourth least-significant-bit of the modulo 80 counter. This counter is public so, for convenience, it is easier to combined the XOR of the counter variable and the key variable into one variable as follows

$$k'_t = k_{t \bmod 80} + c_t^4. \tag{10}$$

Keystream is not produced in initialisation. Instead, the output of the filter function is used to update the state. Output is produced using the following filter function

$$\begin{aligned}z_t = s_{t+30} + b_{t+1} + b_{t+6} + b_{t+15} + b_{t+17} + b_{t+23} + b_{t+28} + b_{t+34} + b_{t+4}s_{t+6}\\ + s_{t+8}s_{t+10} + s_{t+32}s_{t+17} + s_{t+19}s_{t+23} + b_{t+4}s_{t+32}b_{t+38}.\end{aligned} \tag{11}$$

Keystream Generation. At the end of initialisation, the LFSR is increased by one bit and the new stage is loaded with a one. Thus, in keystream generation Plantlet consists of a 40-bit NFSR and a 61-bit LFSR. This adjustment to the LFSR is made in order to avoid the possibility of the LFSR being initialised to the all zero state.

The update functions used to update the state of the LFSR and the NFSR during keystream generation are identical to those used in initialisation; however, the output function is now used to generate keystream and so it is not used to update the state.

5.2 Modified Version of Plantlet

We introduce a modified version of the Plantlet stream cipher, where we replace any independent linear term taken from the NFSR by the corresponding term in the LFSR. That is, the filter function $f(B, S)$ remains the same, except that monomials appearing only in L_B are replaced by monomials in L_S with the same indices. Note that we denote the modified version of Plantlet by appending the suffix $-m$.

Table 1 highlights the differences between the original and modified versions.

Table 1. Modifications to linear combinations in Plantlet.

Original linear combination	Modified linear combination
$b_{t+1} + b_{t+6} + b_{t+15} + b_{t+17} + b_{t+23} + b_{t+28} + b_{t+34}$	$s_{t+1} + s_{t+6} + s_{t+15} + s_{t+17} + s_{t+23} + s_{t+28} + s_{t+34}$

5.3 Stage 1: LFSR Recovery

In this section we apply the algorithm from Sect. 4. The theoretical data and computational complexity requirements to recover the LFSR initial state is summarised in Table 2. In Sect. 6, we provide experimental results for the modified version of Plantlet.

At time $t = 0$ an output bit in Plantlet-m is produced as follows:

$$z_0 = s_1 + s_6 + s_{15} + s_{17} + s_{23} + s_{28} + s_{34} + b_4 s_6 + s_8 s_{10} + s_{32} s_{17} + s_{19} s_{23} + b_4 s_{32} b_{38}$$

Multiplying this equation by $(s_6 + 1)(s_{32} + 1)$ gives

$$\begin{aligned}(s_6 + 1)(s_{32} + 1)z_0 = {} & s_1 s_6 s_{32} + s_1 s_6 + s_1 s_{32} + s_1 + s_6 s_{15} s_{32} + s_6 s_{15} + s_6 s_{17} s_{32} \\ & + s_6 s_{17} + s_6 s_{23} s_{32} s_{19} + s_6 s_{23} s_{32} + s_6 s_{23} s_{19} + s_6 s_{23} \\ & + s_6 s_{28} s_{32} + s_6 s_{28} + s_6 s_{34} s_{32} + s_6 s_{34} + s_6 s_8 s_{10} s_{32} \\ & + s_6 s_8 s_{10} + s_{15} s_{32} + s_{15} + s_{17} s_{32} + s_{17} + s_{23} s_{32} s_{19} \\ & + s_{23} s_{32} + s_{23} s_{19} + s_{23} + s_{28} s_{32} + s_{28} + s_{34} s_{32} \\ & + s_{34} + s_8 s_{10} s_{32} + s_8 s_{10}\end{aligned} \tag{12}$$

where the right hand side of the equation contains only LFSR initial state bits and is of degree 4. Thus, by observing at least $\binom{61}{4}$ keystream bits, fast algebraic techniques may be applied in the precomputation phase of the attack to reduce the overall degree of the system to the degree of the left hand side (which is of degree 2 in the unknown LFSR initial state bits) [6].

Table 2. Resource requirements for recovering the LFSR of modified Plantlet.

Precomputation phase	
Degree before fast algebraic techniques	4
Complexity	$\mathcal{O}(2^{23})$
Degree after fast algebraic techniques	2
Online phase	
Data	2^{19}
Complexity	$\mathcal{O}(2^{32})$

5.4 Stage 2: NFSR Recovery and Key Recovery

Once the LFSR initial state is recovered, the output function will contain only unknown state bits from the NFSR. As described in Sect. 4.1, to be able to predict the NFSR state sequence we must know at least $|NFSR| + |K|$ consecutive bits of the NFSR sequence. For Plantlet, at least $40 + 80 = 120$ consecutive bits of NFSR sequence is required to predict the NFSR sequence. Note that if 120 bits of NFSR sequence is known, the key can easily be calculated. Thus, the goal is to recover 120 bits of consecutive NFSR bits.

The data requirement for this stage will utilise the data collected for LFSR state recovery. The computational complexity to partially recover the NFSR sequence is considered to be negligible [4]. The number of NFSR sequence bits recovered over a 120-bit period through application of this method is hard to estimate and will vary depending on the particular key and IV used. However, some guidance based on experimental results is provided in Sect. 6.2. Due to the low computational complexity of partial NFSR sequence recovery we provide experimental results for this in the next section.

At time $t = 0$ an output bit in Plantlet-m is produced by:

$$z_0 = s_{t+1} + s_{t+6} + s_{t+15} + s_{t+17} + s_{t+23} + s_{t+28} + s_{t+34} + b_{t+4}s_{t+6} \\ + s_{t+8}s_{t+10} + s_{t+32}s_{t+17} + s_{t+19}s_{t+23} + b_{t+4}s_{t+32}b_{t+38}$$

This function is linear in the NFSR bit b_4 and nonlinear in the NFSR bits b_4 and b_{38}. At each time step we have:

$$z_t = \alpha b_{4+t} + \beta b_{4+t}b_{38+t} + \omega,$$

where α, β and ω can be 0 or 1, respectively.

When $\alpha = 0$, $\beta = 1$, and $\omega + z = 1$, two NFSR sequence bit will be recovered. When $\alpha = 1$ and $\beta = 0$, an NFSR initial state bit will be recovered. Finally, when $\alpha = 1$, $\beta = 1$, and $\omega + z = 1$, two NFSR sequence bits will be recovered.

This can be used for simple partial state recovery of the NFSR. The remaining stages of the 120 NFSR sequence can then be found through exhaustive search. An estimate of the average exhaustive search requirement for modified Plantlet is provided in Table 3 of Sect. 6.2.

6 Experimental Simulations

We have performed computer simulations of our divide and conquer attack, applying it to our modified version of Plantlet, to demonstrate proof of concept. The details of the simulation setup and results are provided in the following sections. We also provide experimental results in Sect. 6.2 for the partial NFSR sequence recovery of Plantlet; this is possible because of the low time complexity required to partially recover. The details for the structure of modified Plantlet are provided in Sect. 5.

6.1 Experimental Approach

For each simulation, a random key together with random NFSR and LFSR states were produced. Output from modified Plantlet was then produced. The attack from Sect. 4 was then applied. The remaining NFSR sequence bits were then exhaustively searched. Each sequence candidate was used to produce output, which was checked against the correct output sequence. A candidate that produced the correct output was considered the correct state sequence state. Using this sequence, the key was recovered. The computed key, NFSR and LFSR was then checked against the correct key, NFSR and LFSR.

The code used for the simulations was written using the SageMath software package [25] and all calculations were performed using QUT's High Performance Computing facility. We used a single node from the cluster with an Intel Xeon core capable of 271 TeraFlops.

6.2 Results on Modified Plantlet

In precomputation, the initial system of equations was built, the linear dependency was found and the reduced system of equations was built. For modified Plantlet, approximately 2^{20} bits of output were used. The majority of the computational complexity required for the precomputation comes from applying the linear dependency to produce the reduced system of equations. On average, precomputation was completed in 10 h.

A total of 10 simulations were performed. In every simulation the full LFSR initial state was recovered. Each simulation for the modified version required on average 30 s to recover the LFSR initial state.

For each trial, partially recovering the required 120-bit NFSR sequence took approximately 2.5 h. Table 3 provides a tally (across the 10 simulations) of how many times a certain number of state bits were recovered from the NFSR sequence. For each simulation, the full available keystream was used. That is, the NFSR sequence was partially recovered using 2^{19} bits of keystream. We see from Table 3 that on average, 67 bits were recovered for the NFSR sequence.

Table 3. Distribution table for NFSR sequence bits recovered over 120-bit windows using 100 simulations for modified Plantlet.

No. bits recovered	**0**	...	**64**	**65**	**66**	**67**	**68**	**69**	...	**120**
Frequency	0	...	0	1	4	2	2	1	...	0

The remaining 53 bits required to complete the 120-bit NFSR sequence could then be recovered by exhaustive search and used to produce output. Recovery of these remaining NFSR bits would then allow the key to be determined. This portion of the simulation was not performed due to limited resources.

7 Discussion

Our experimental simulations support the theoretical model for our key recovery attack on modified Plantlet. The pre-computation stage of the algebraic attack is essential for recovery of the LFSR state and is the most time-consuming part of the process, but only needs to be done once. This took ten hours in our trial. In the online phase, a divide and conquer approach targeting the LFSR achieved complete recovery of the LFSR state in approximately 30 s with 100% success. Following this, we used a 120-bit sliding window on the NFSR state and partially recovered this window; on average we recovered 67 of these 120 bits. Guessing the remaining 53 bits and checking for consistency with observed keystream then allowed us to recover the key bits. The complexity of this guess and determine stage dominates the attack complexity but is clearly significantly less complex than exhaustive key search.

The use of key injection in the design made it possible to perform this key recovery attack without requiring any consideration of the initialisation process. That is, the initialisation is irrelevant to the security provision against these algebraic attacks. In fact, security against these attacks depends solely on the combination of the selected output function and the positions of its input bits within the two registers. This answers the open question from [1] of whether key injection provides increased security to enable reduced register sizes for lightweight stream cipher designs. While this approach may help avoid time-memory trade-off attacks, the generic structure is not robust: other attacks are possible, as we have shown.

Based on the nature of our successful attack on modified Plantlet, the following design guideline is seen to be important to the security of any keystream generator which uses a Grain-like structure with key injection:

- The output function for any such cipher should contain multiple bits taken linearly from the NFSR, with none of these bits involved in nonlinear terms of the output function that also contain bits from the LFSR.

Note that this is precisely the feature which protects the existing version of Plantlet from our attack. This guideline may need to be expanded if other types of attack on this structure are also found to be successful.

The output function of Sprout is identical to that of Plantlet, so the published version of Sprout is also protected against our attack. However, a similar modification of this output function would again allow our attack to recover the initial contents of Sprout's LFSR. But recovering the NFSR initial state and the key is more complex for Sprout, since the key bits of Sprout are fed into the NFSR conditionally. A quick estimate based on our experimental results for Plantlet suggests that the exhaustive search cost after partial NFSR recovery in this case would exceed the cost of exhaustively searching the key bits, making this attack unviable.

8 Conclusion

In this paper, we have considered the security of keystream generators using a Grain-like structure with key injection. From the above discussion, it is clear that the security of keystream generators using this design depends critically on the choice of the output function during keystream generation. Designers who employ this design approach should pay careful attention to the combination of the function used and the location of the input taps which feed it.

Ciphers of this type were designed specifically to avoid time-memory tradeoff attacks, but they are not necessarily secure against other emerging attacks, such as the algebraic attack we have demonstrated here. In itself, this structure cannot be considered to provide a robust generic design for lightweight keystream generators. Indeed, the designer of a new cipher must be cautious about security implications when adding components to existing structures and should evaluate the modified design against all possible types of attacks.

A Appendix: Algorithms

A.1 Algorithm for NLFG Algebraic Attack

Precomputation phase:

Step 1. Use $f(S_0) = y_0$ to relate initial state bits $(s_0, s_1, \ldots, s_{n-1})$ to observed output bit y_0.
Step 2. Multiply f by a function h (if applicable) to reduce overall degree to d.
Step 3. Clock forward using $f(S_t) = y_t$ to build a system of equations of constant algebraic degree, applying the linear update as required.

Online phase:

Step 4. Substitute observed output bits $\{y_t\}_{t=0}^{\infty}$ into the system of equations.
Step 5. Solve the system of equations by linearisation, to recover $S_0 = s_0, s_1, \ldots, s_{n-1}$.

In the online phase of this attack, the initial state of the LFSR can be recovered if approximately $\binom{n}{d}$ bits of output are known. The attack has a computational complexity of $\mathcal{O}(n\binom{n}{d} + \binom{n}{d}^{\omega})$, where d is the degree of the system and ω is the Guassian elimination exponent $\omega \approx 2.8$ [7]. If the output requirement cannot be met, it may be possible to solve the system by applying other methods for solving simultaneous equations, such as Gröbner bases or the XL algorithm [5].

A.2 Algorithm for Fast Algebraic Attack

The precomputation phase is similar to a regular algebraic attack, with Step 3 replaced by three steps (3a, 3b and 3c) as follows.

Step 3a. Identify the combination of equations that will eliminate monomials of degree e to d in the initial state bits.
Step 3b. Use this linear dependency to build a new general equation.
Step 3c. Use this general equation to build a system of equations of degree e in the initial state bits.

The online phase is identical to the online phase of a regular algebraic attack (but with reduced complexity).

When the Berlekamp-Massey algorithm is used to find the linear dependency, the pre-computation phase of the attack has a computational complexity of $\mathcal{O}(\binom{n}{d} log(\binom{n}{d})))$ [6]. The initial state of the LFSR can be recovered in the online phase of the attack by observing approximately $\binom{n}{d}$ bits of output with a computational complexity of $\mathcal{O}(\binom{n}{d}\binom{n}{e} + \binom{n}{e}^{\omega})$, where d is the degree of fh, e is the degree of h and $\omega \approx 2.8$ [6].

Note that at first glance the online complexities for an algebraic attack and a fast algebraic attack look similar. However, when n is much larger than d, as is the case with registers used in practice, $\binom{n}{d}^{\omega}$ is much larger than $\binom{n}{d}\binom{n}{e}$ and $\binom{n}{e}^{\omega}$. Thus, by reducing the degree from d to e, the complexity of the online phase is drastically reduced for registers of practical size.

A.3 Algorithm for LF-NFSR Algebraic Attack

Precomputation phase:

Step 1. A system of equations is developed using the linear filter function to represent every state bit as a linear combination of a subset of the initial state bits and some output bits. We denote this system of equation by system $\mathcal{L}$.

Step 2. A second system of equations is developed using the nonlinear update function g to represent update bits as a nonlinear combination of a subset of initial state bits. We denote this system by system $\mathcal{G}$. Substitutions are made for state bits in system $\mathcal{G}$ using system $\mathcal{L}$ where applicable to reduce the number of unknown state variables while keeping the degree of system $\mathcal{G}$ constant.

Step 3. The two systems are combined by aligning the equations from each system that represent the same state bit. The resulting system contains only initial state bits and observed output bits. We denote this system as system $\mathcal{L} + \mathcal{G}$.

Online phase:

Step 4. Substitute observed output bits $\{y_t\}_{t=0}^{\infty}$ into the system of equations

Step 5. Solve the system of equations by linearisation.

For certain update functions a reduction function of g, say h, may be used to reduce the overall degree of the system. If the degree of gh is d, then the overall system will be of degree at most d. The initial state of the LF-NFSR can be recovered in the online phase of the attack by observing approximately $\binom{n}{d}$ bits of output with a computational complexity of $\mathcal{O}(n\binom{n}{d} + \binom{n}{d}^{\omega})$, where d is the degree of the system and $\omega \approx 2.8$ [3]. Note that fast algebraic techniques are not applicable to LF-NFSRs.

References

1. Armknecht, F., Mikhalev, V.: On lightweight stream ciphers with shorter internal states. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 451–470. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_22

2. Beighton, M., Bartlett, H., Simpson, L., Wong, K.K.H.: Algebraic attacks on Grain-like keystream generators. In: Park, J.H., Seo, S.H. (eds.) ICISC 2021. LNCS, vol. 13218, pp. 241–270. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08896-4_12
3. Berbain, C., Gilbert, H., Joux, A.: Algebraic and correlation attacks against linearly filtered non linear feedback shift registers. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 184–198. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_12
4. Berbain, C., Gilbert, H., Maximov, A.: Cryptanalysis of Grain. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 15–29. Springer, Heidelberg (2006). https://doi.org/10.1007/11799313_2
5. Courtois, N.T.: Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 182–199. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36552-4_13
6. Courtois, N.T.: Fast algebraic attacks on stream ciphers with linear feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_11
7. Courtois, N.T., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_21
8. Englund, H., Johansson, T.: A new simple technique to attack filter generators and related ciphers. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 39–53. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30564-4_3
9. Faugere, J.C., Ars, G.: An algebraic cryptanalysis of nonlinear filter generators using Gröbner bases. Report, INRIA (2003)
10. Forré, R.: A fast correlation attack on nonlinearly feedforward filtered shift-register sequences. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 586–595. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-46885-4_56
11. Gammel, B.M., Göttfert, R.: Linear filtering of nonlinear shift-register sequences. In: Ytrehus, Ø. (ed.) WCC 2005. LNCS, vol. 3969, pp. 354–370. Springer, Heidelberg (2006). https://doi.org/10.1007/11779360_28
12. Golić, J.D., Salmasizadeh, M., Simpson, L., Dawson, E.: Fast correlation attacks on nonlinear filter generators. Inf. Process. Lett. **64**(1), 37–42 (1997)
13. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 179–190. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_14
14. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: 2006 IEEE International Symposium on Information Theory, pp. 1614–1618. IEEE (2006)
15. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. Int. J. Wirel. Mob. Comput. **2**(1), 86–93 (2005)
16. Hell, M., Johansson, T., Meier, W.: Grain-128a: a new version of Grain-128 with optional authentication. Int. J. Wirel. Mob. Comput. **5**, 48–59 (2011)
17. Hell, M., Johansson, T., Meier, W., Sönnerup, J., Yoshida, H.: Grain-128AEAD - a lightweight AEAD stream cipher. NIST Lightweight Cryptography Competition (2019). https://csrc.nist.gov/Projects/lightweight-cryptography/finalists

18. Hong, J., Sarkar, P.: New applications of time memory data tradeoffs. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 353–372. Springer, Heidelberg (2005). https://doi.org/10.1007/11593447_19
19. Katz, J., Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)
20. Massey, J.: Shift-register synthesis and BCH decoding. IEEE Trans. Inf. Theory **15**(1), 122–127 (1969)
21. Meier, W., Staffelbach, O.: Fast correlation attacks on certain stream ciphers. J. Cryptol. **1**(3), 159–176 (1989)
22. Mikhalev, V., Armknecht, F., Müller, C.: On ciphers that continuously access the non-volatile key. IACR Trans. Symmetric Cryptol. 52–79 (2016)
23. Millan, W.: Analysis and design of Boolean functions for cryptographic applications. Ph.D. Thesis, Queensland University of Technology (1997)
24. Siegenthaler, T.: Cryptanalysts representation of nonlinearly filtered ML-sequences. In: Pichler, F. (ed.) EUROCRYPT 1985. LNCS, vol. 219, pp. 103–110. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39805-8_12
25. Stein, W., Joyner, D.: Sage: system for algebra and geometry experimentation. ACM Bull. **39**(2), 61–64 (2005)

Related-Cipher Attacks: Applications to Ballet and ANT

Yongxia Mao[1,2,3], Wenling Wu[1,3](✉), Yafei Zheng[1,2,3], and Lei Zhang[1,3]

[1] Trusted Computing and Information Assurance Laboratory, Institute of Software
Chinese Academy of Sciences, Beijing 100190, China
{yongxia2018,wenling,zhengyafei,zhanglei}@iscas.ac.cn
[2] State Key Laboratory of Cryptology, Beijing 100878, China
[3] University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. Quite a lot of block ciphers proposed in recent years are families of ciphers that conveniently support multiple block lengths and key lengths. The essential security requirements for a family of block ciphers are: (1) Each cipher instance from family is secure; (2) Cipher instances do not endanger each other's security, namely, by one or more cipher instances, other instances cannot be predicted. However, traditional cryptanalysis methods always assess the security of a special member of the family cipher, such as differential cryptanalysis, linear cryptanalysis, integral cryptanalysis. Related-cipher attacks focus on the security between cipher instances. This paper researches the security of Ballet-128 and ANT-128 against related-cipher attacks. Since their key schedules do not rely on the round number of encryption, we consider the related-cipher attack with equivalent keys by limiting the 256-bit key space. As a result, we recover the secret key of the full Ballet-128/128 with just one chosen plaintext pairs and one call of Ballet-128/128 and Ballet-128/256, which means Ballet-128 is insecure against related-cipher attack. For ANT-128, we show that there exist at most 6-round related-cipher distinguishers between ANT-128/128 and ANT-128/256, and launch a 9-round key-recovery attack on ANT-128/128 based on a 6-round related-cipher distinguisher with the time complexity about $2^{60.9}$.

Keywords: Related-cipher attack · Block cipher family · Key schedule · Key recovery

1 Introduction

In recent years, a family of block ciphers that support multiple block lengths and key lengths is a major feature of the newly designed block cipher. For example, KATAN and KTANTAN [1], Simon and Speck [2], Simeck [3], and LowMC [4] When evaluating the security of a cryptographic algorithm family, the evaluator often only consider the security of a single algorithm member.

The essential security requirements for a family of block ciphers are: (1) Each cipher instance of family is secure; (2) Cipher instances do not endanger

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 109–123, 2023.
https://doi.org/10.1007/978-3-031-35486-1_6

each other's security, namely, by one or more cipher instances, neither the corresponding key can be recovered, nor other instances can be predicted. Differential analysis [5], linear analysis [6], impossible differential [7] and integral analysis [8] etc. are all aimed at the first security requirement of the cipher family. Attacks focusing on the second security requirement of the cipher family are collectively referred to as *related-cipher attacks*.

Differential analysis, linear analysis, impossible differential analysis, integral analysis and meet-in-the-middle analysis [9] are traditional cryptanalysis methods for block ciphers. In terms of evaluating a cipher family, we always pay attention to the security of each algorithm instance, and regard the single member's security as the ability of entire cipher family against these attacks. Obviously, it is insufficient. When considering the application environments, the security requirements of cipher families will be higher. For example, when developing cryptographic products, developers usually implant the entire algorithm family into the chip and call a specific one when using them; or for designing a module that integrates multiple functions, the same underlying algorithm is used. Therefore, the security of algorithms may affect each other. If we know one of algorithms' secret key, the other algorithms' or functions' key may be predicted correctly in the system. Under these circumstances, the whole system's security is not equivalent to the security of individual primitive, and we need to further research the security impact between cryptographic algorithms. Related-cipher attacks and related-key attacks [10] are right analysis methods focusing on the security impact between cipher instances. Therefore, it is particularly necessary to evaluate related-cipher attacks on the cipher families.

The early block cipher AES [11] and SQUARE [12] are also ciphers with multiple block lengths and key sizes. Although many related research results on AES have been published [13–15], most of them focus on the security analysis of AES-128/128, a specific instance of AES. In 2002, Wu Hongjun [16] proposed an attack on related ciphers of AES-128/128 and AES-128/256 under the same key, and pointed out a new AES key schedule proposed at ACISP 2002 is weaker than the original one under this attack. In 2005, Jaechul Sung and others [17] extended the key form of related-cipher attacks to "semi-equivalent keys", and applied this attack to the block ciphers ARIA, SC2000. Since typical related-cipher attack is more difficult to apply as the difference of round numbers becomes bigger, they combined related-cipher attack with differential cryptanalysis, linear cryptanalysis, higher-order differential cryptanalysis and so on, to attack SAFER++, CAST-128 and DEAL. More recently, Shao Zengyu and Ding Lin [18] applied related-cipher attack to the stream cipher Salsa20, and claimed that if a secret key is used in Salsa20/12 and Salsa20/8, 256-bit secret keys can be recovered with a time complexity of 2^{224}, and then Ding Lin [19] further reduced the cost of this attack by changing the initial IV and repeatedly using related-cipher attacks. This attack also can be applied to protocols. In 2004, Kohno [20] used this attack to analysis WinZIP's encryption scheme, and found that the encryption way may leak information about the encrypted files when changing the cipher used in WinZip archives.

Ballet [21] and ANT [22] are the participating ciphers in the National Cryptographic Algorithm Design Competition (NCADC) [23] organized by the Chinese Association for Cryptologic Research in 2018. Their block lengths/key lengths support 128/128, 128/256, 256/256. During the competition, evaluators evaluated the security of Ballet and ANT. For Ballet-128, with the help of automated modeling technology, the effective differential paths do not exceed 26 rounds, the effective linear characteristics do not exceed 25 rounds, the longest impossible differential path and integral distinguishers are both 7 rounds, and the longest zero-correlation linear paths is 6 rounds. With the help of SAT/SMT technique, ANT-128 does not have effective differential and linear characteristics for more than 27 rounds, does not have integral distinguishers for more than 16 rounds by searching bit-level division trails, and only have impossible differential routes for up to 9 rounds and zero-correlation linear paths for up to 10 rounds can be retrieved. The evaluation result have shown that the two cipher algorithms are secure enough against general cryptanalysis methods.

Our Contributions. In this paper, we research the security of Ballet and ANT against related-cipher attacks. At first, for Ballet-128/128 and Ballet-128/256, which are two family members of Ballet, we capture a property (Property 1) that their round keys satisfy a relation since their key schedules are similar and independent on total round numbers. Depending on the relation, we find a high-round distinguisher between Ballet-128/128 and Ballet-128/256, and then launch a key-recovery attack on full Ballet-128/128. As a result, Ballet-128 is insecure against related-cipher attack. Then, for ANT-128/128 and ANT-128/256, which are two family members of ANT, we present a 6-round distinguisher between them, launch a 9-round key-recovery attack on ANT-128/128 based on the 6-round distinguisher, and show that the non-existence of 7-round related-cipher distinguishers under the condition of equivalent keys.

Organization. The rest of the paper is organized as follows. Section 2 introduces related-cipher attacks and block ciphers we will attack. Section 3 presents the related-cipher attack against Ballet-128. Section 4 evaluates the security of ANT-128 under the related-cipher attack. Finally, the paper ends in Sect. 5 with a conclusion.

2 Preliminaries

In this section, we are going to recall related-cipher attack with taking SQUARE as an example in [16], and then briefly present our target block ciphers.

2.1 Related-Cipher Attack

In 2002, Wu Hongjun [16] formally introduced the concept of related-cipher attack, and considered the related ciphers as block ciphers with the same round function but with different round numbers. Consider two related block ciphers with different round numbers, r and $(r + \Delta r)$ respectively. If a key is used in these two ciphers to encrypt the same message, the attack can be carried out on the Δr-round cipher. For this Δr-round cipher, the plaintext is the ciphertext of the r-round cipher and the ciphertext is that of the $(r + \Delta r)$-round cipher. The key can be determined easily for small Δr. Taking the attack on SQUARE with flexible round number as an example.

SQUARE is a iterative block cipher. As a safety margin, the designers fixed the number of rounds to eight. However, the designers also allow conservative users to increase the number of rounds in a straight way. Let SQUARE^r and SQUARE$^{r+\Delta r}$ be two block ciphers with encrypting r rounds and $(r + \Delta r)$ rounds.

Denote the ith round function of SQUARE as: $\rho[k^i] = \sigma[k^i] \circ \pi \circ \gamma \circ \theta$, where k^i is the ith round key, θ is a linear transformation, γ is a nonlinear transformation, and π is a byte permutaion. Let c^r be the ciphertext of r-round SQUARE and $c^{r+\Delta r}$ be that of $(r + \Delta r)$-round SQUARE. If c^r and $c^{r+\Delta r}$ are related to the same plaintext P, that is, $c^r = \rho^r(P)$ and $c^{r+\Delta r} = \rho^{r+\Delta r}(P)$, $(c^r, c^{r+\Delta r})$ is denoted as one right pair. In this case, the related-cipher attack can be applied when $\Delta r = 1$ and $\Delta r = 2$.

For two encryption algorithms, SQUARE^r and SQUARE$^{r+\Delta r}$, the attacker randomly choose a plaintext P, and he can know the ciphertext c^r and $c^{r+\Delta r}$ by encrypting plaintext P. It means the $(r+1)$-round input and $(r+\Delta r)$-round output of SQUARE$^{r+\Delta r}$ are known. When $\Delta r = 1$, SQUARE$^{r+\Delta r}$ is reduced to only one round and the relation $c^{r+1} = \rho[k^{r+1}](c^r)$ holds. Hence, the round key k^{r+1} can be calculated by

$$k^{r+1} = (\pi \circ \gamma \circ \theta(c^r)) \oplus c^{r+1}.$$

Furthermore, other round keys of SQUARE can be deduced due to $k^i = g(k^{i-1})$, where g is the key schedule. The above attack only needs one right pair. Similarly, if $\Delta r = 2$, the round key can be determined easily from two right pairs (c^r, c^{r+2}) by the relation

$$c^{r+2} = \rho[k^{r+2}] \circ \rho[k^{r+1}](c^r).$$

Therefore, SQUARE is insecure against related-cipher attack.

From the level of ciphers, related-cipher attack concentrates on whether there is a potential relationship between cipher instances which contributes to leaking key information or state information. This is exactly captured and contained in the description of *related-cipher attack* in the introduction. To some extent, related-key attack and multi-key attack [24] can be also classified into the category of related-cipher attack, because they both research the impact of security between cipher instances where the encryption algorithms are fixed.

2.2 Target Block Ciphers

In 2018, the National Cryptographic Algorithm Design Competition was organized by the Chinese Association for Cryptologic Research, and 22 block ciphers totally entered the first round of candidates. A common feature of them is that they all support at least three block lengths and key lengths. Finally, uBlock [25] and Ballet won the first prize with their excellent security and great performances. ANT and other two ciphers won the second prize.

Ballet. Ballet is an ARX block cipher, and supports Ballet-128/128, Ballet-128/256, Ballet-256/256 versions. The round function consists of cyclic shift, modular addition and XOR operation, as shown in Fig. 1, where $(X_0^i||X_1^i||X_2^i||X_3^i)$ denotes the *ith* round input and $(sk_i^L||sk_i^R)$ is round keys, $i = 0, 1, \cdots, r-1$. Denote the round function of Ballet as $F = \pi \circ \tau \circ \rho$, where ρ is the combination of rotation, modular addition and addition, τ is the round key addition, and π is the vector permutation. The details of these transformations are shown in Fig. 1. The last round function omits the linear permutation operation. Encryption rounds of Ballet-128/128 and Ballet-128/256 are 46 and 48 respectively. Specific key schedules are shown in Table 1, where $(rk_i^L||rk_i^R)$ denotes the round key of Ballet-128/256, and X_*^i, sk_i^* , $rk_i^* \in \mathbb{F}_2^{32}$.

Table 1. Key Schedule of Ballet-128

Key Schedule of Ballet-128/128	Key Schedule of Ballet-128/256
Master key $K = k_0\|\|k_1$	Master key $K = k_0\|\|k_1\|\|t_0\|\|t_1$
For $0 \leq i < r$;	For $0 \leq i < r$;
$(sk_i^L\|\|sk_i^R) = k_0$;	$(rk_i^L\|\|rk_i^R) = k_0$;
Output $(sk_i^L\|\|sk_i^R)$;	Output $(rk_i^L\|\|rk_i^R)$;
$k_{temp} = k_1$;	$t_{temp} = t_1$;
$k_1 = k_0 \oplus (k_1 <<< 3) \oplus (k_1 <<< 5) \oplus i$;	$t_1 = t_0 \oplus (t_1 <<< 7) \oplus (t_1 <<< 17)$;
$k_0 = k_{temp}$	$t_0 = t_{temp}$;
	$k_{temp} = k_1$;
	$k_1 = k_0 \oplus (k_1 <<< 3) \oplus (k_1 <<< 5)$;
	$k_0 = k_{temp}$;
	$k_1 = k_1 \oplus t_1 \oplus i$

ANT. ANT is a Feistel-network block cipher, and supports ANT-128/128, ANT-128/256, ANT-256/256 versions. The round function F_{sk_i} is composed of AND operation, rotation and XOR operation, as shown in Fig. 2, and can be defined as

$$(L_{i+1}, R_{i+1}) = F_{sk_i}(L_i, R_i) = (G_0(L_i <<< 3) \oplus G_1(L_i <<< 16) \oplus R_i \oplus sk_i, L_i)$$

where $0 \leq i < r$, $(3, 16)$ are rotation parameters, G_0 and G_1 are nonlinear functions that contain two layers, with a bit level of permutation in between. Before introducing the nonlinear functions G_0 and G_1, some representations are introduced as follows:

Let $n = 64$. Denote $x^0 = x_{n-1}^0||x_{n-2}^0||\cdots||x_0^0$ as the input of G_0 or G_1, and $y^0 = y_{n-1}^0||y_{n-2}^0||\cdots||y_0^0$ as the first layer output of G_0 or G_1. Similarly, denote $x^1 = x^1_{n-1}||x^1_{n-2}||\cdots||x^1_0$ and $y^1 = y^1_{n-1}||y^1_{n-2}||\cdots||y^1_0$ as the second layer input and output of G_0 or G_1, respectively. $PERM$ represents the bit level of permutation in G_0 or G_1. The symbol $\odot$ represents the operation AND. A specific description of G_0 and G_1 is given below.

For G_0: $y^1 = G_0(x^0)$, where the first layer transformation is

$$y^0_j = \begin{cases} (x^0_{j+3} \odot x^0_{j+2}) \oplus x^0_j, j \, mod\, 4 = 0, \\ x^0_j, \quad others. \end{cases} for\, 0 \leq j < n.$$

the bit permutation is

$$x^1_{PERM(j)} = y^0_j, \quad for\, 0 \leq j < n.$$

and the second layer transformation is

$$y^1_j = \begin{cases} (x^1_{j+3} \odot x^1_{j+2}) \oplus x^1_j, j \, mod\, 4 = 0, \\ x^1_j, \quad others. \end{cases} for\, 0 \leq j < n.$$

Similarly, for $G_1 : y^1 = G_1(x^0)$, where the first layer transformation is

$$y^0_j = \begin{cases} (x^0_{j+2} \odot x^0_{j+1}) \oplus x^0_j, j \, mod\, 4 = 1, \\ x^0_j, \quad others. \end{cases} for\, 0 \leq j < n.$$

the bit permutation is

$$x^1_{PERM(j)} = y^0_j, \quad for\, 0 \leq j < n.$$

and the second layer transformation is

$$y^1_j = \begin{cases} (x^1_{j+2} \odot x^1_{j+1}) \oplus x^1_j, j \, mod\, 4 = 1, \\ x^1_j, \quad others. \end{cases} for\, 0 \leq j < n.$$

The total round number of ANT-128/128 and ANT-128/256 are 56 and 70, respectively. Denote $2n$ and $4n$ as cipher key sizes of ANT, and key schedules can be described as follows.

For ANT-$2n/2n$, cipher key $K = k_{2n-1}||\cdots||k_0$ can be divided into two words:

$$K_1 = k_{2n-1}||\cdots||k_n, \; K_0 = k_{n-1}||\cdots||k_0.$$

$K_1||K_0$ works as the initial state of LFSR in Fig. 4. For different block sizes, inputs of operation A (as shown in Fig. 3) will be divided into 8 $n/8$-bit vectors $X_7||\cdots||X_0$. During each update of the LFSR, operation A is applied to K_{i+1} for 3 times. The n-bit K_i in the register will be used as the current round key sk_i. Rotation parameters are $(t_0, t_1) = (7, 1)$.

For ANT-$2n/4n$, master key $K = k_{4n-1}||\cdots||k_0$ can be divided into four words:

$$K_3 = k_{4n-1}||\cdots||k_{3n}, \; K_2 = k_{3n-1}||\cdots||k_{2n},$$
$$K_1 = k_{2n-1}||\cdots||k_n, \; K_0 = k_{n-1}||\cdots||k_0.$$

$K_3||K_2||K_1||K_0$ works as the initial state of LFSR in Fig. 5. During each update of the LFSR, operation A is applied to K_{i+1} for 3 times. Rotation parameters are $(t_0, t_1) = (7, 1)$.

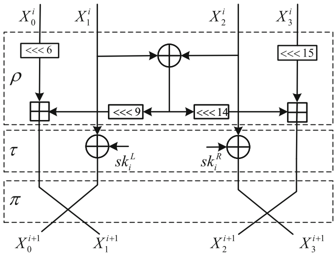


Fig. 1. Round function of Ballet

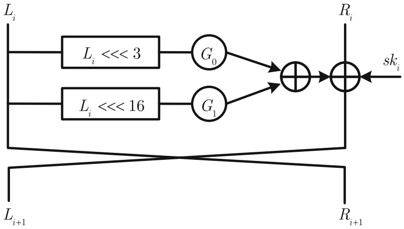


Fig. 2. Round function of ANT

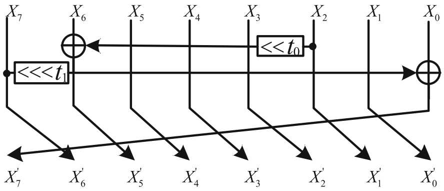


Fig. 3. operation A

Let the cipher key be $K_{128} = k_0||k_1$ and $K_{256} = K_3||K_2||K_1||K_0$ for ANT-128/128 and ANT-128/256 respectively. The ith round subkey are sk_i and rk_i, $0 \leq i < r$. Then, we can get

$$
\begin{aligned}
& sk_0 = k_0,\ sk_1 = k_1, \\
for\ i \geq 2,\ & sk_i = sk_{i-2} \oplus (i-1) \oplus A^3(sk_{i-1}), \\
& rk_0 = K_0,\ rk_1 = K_1,\ rk_2 = K_2,\ rk_3 = K_3, \\
for\ j \geq 4,\ & rk_j = rk_{j-4} \oplus (j-3) \oplus rk_{j-3} \oplus A^3(rk_{j-1}).
\end{aligned}
$$

3 Related-Cipher Attack on Ballet

3.1 Property of Ballet-128

Ballet-128 denotes the family member of Ballet with 128-bit block length, so it contains Ballet-128/128 and Ballet-128/256. As shown in Sect. 2, key schedules of Ballet-128 has a common construction if we regard the role of t_i generated by the branch as an operation of a simple parameter table computed in advance. Thus, we have the following property.

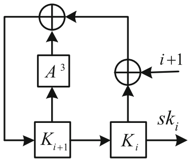


Fig. 4. Key schedule of ANT-$2n/2n$

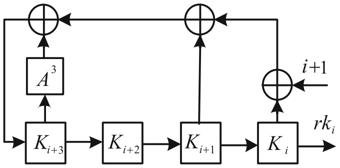


Fig. 5. Key schedule of ANT-$2n/4n$

Property 1. Let $K_{128} = k_0||k_1$ denote the 128-bit master key of Ballet-128/128, sk_i and rk_i, $0 \le i < r$, denote the *ith* round key of Ballet-128/128 and Ballet-128/256 respectively. If the 256-bit master key of Ballet-128/256 is $K_{256} = k_0||k_1||t_0||t_1$, $t_0,\ t_1 \in \{(0\cdots0),(01\cdots01),(10\cdots10),(1\cdots1)\}$, then for $j \ge 0$, round keys of Ballet-128/128 and Ballet-128/256 satisfy the following relation:

$$rk_{4j} = sk_{4j}, rk_{4j+1} = sk_{4j+1},\ rk_{4j+2} = sk_{4j+2} \oplus t_0, rk_{4j+3} = sk_{4j+3} \oplus t_1. \quad (1)$$

Proof. Let $f(x) = (x <<< 3) \oplus (x <<< 5)$, $g(x) = (x <<< 7) \oplus (x <<< 17)$. For $t_0,\ t_1 \in \{(0\cdots0),(01\cdots01),(10\cdots10),(1\cdots1)\}$, $f(t_0) = g(t_0) = 0$, and $f(t_1) = g(t_1) = 0$.

According to the key schedule, when $j = 0$, we have

$$\begin{aligned}
&rk_0 = k_0 = sk_0,\\
&rk_1 = k_1 = sk_1,\\
&sk_2 = sk_0 \oplus f(sk_1) \oplus 0,\\
&rk_2 = rk_0 \oplus f(rk_1) \oplus 0 \oplus t_0 = sk_2 \oplus t_0,\\
&sk_3 = sk_1 \oplus f(sk_2) \oplus 1,\\
&rk_3 = rk_1 \oplus f(rk_2) \oplus 1 \oplus t_1 = sk_1 \oplus f(sk_2) \oplus 1 \oplus t_1 = sk_3 \oplus t_1.
\end{aligned}$$

Suppose that, when $0 \le j < m$,

$$\begin{aligned}
&rk_{4j} = sk_{4j}, rk_{4j+1} = sk_{4j+1},\\
&rk_{4j+2} = sk_{4j+2} \oplus t_0, rk_{4j+3} = sk_{4j+3} \oplus t_1.
\end{aligned}$$

are all satisfied. It can be proved that, when $j = m$, Eq. (1) holds.

$$\begin{aligned}
rk_{4m} &= rk_{4m-2} \oplus f(rk_{4m-1}) \oplus (4m-2) \oplus t_0\\
&= sk_{4m-2} \oplus t_0 \oplus f(sk_{4m-1} \oplus t_1) \oplus (4m-2) \oplus t_0\\
&= sk_{4m-2} \oplus f(sk_{4m-1}) \oplus (4m-2) = sk_{4m},\\
rk_{4m+1} &= rk_{4m-1} \oplus f(rk_{4m}) \oplus (4m-1) \oplus t_1\\
&= sk_{4m-1} \oplus t_1 \oplus f(sk_{4m}) \oplus (4m-1) \oplus t_1 = sk_{4m+1},\\
rk_{4m+2} &= rk_{4m} \oplus f(rk_{4m+1}) \oplus 4m \oplus t_0\\
&= sk_{4m} \oplus f(sk_{4m+1}) \oplus 4m \oplus t_0 = sk_{4m+2} \oplus t_0,\\
rk_{4m+3} &= rk_{4m+1} \oplus f(rk_{4m+2}) \oplus (4m+1) \oplus t_1\\
&= sk_{4m+1} \oplus f(sk_{4m+2} \oplus t_0) \oplus (4m+1) \oplus t_1 = sk_{4m+3} \oplus t_1.
\end{aligned}$$

In summary, for any $j \ge 0$, Eq. (1) holds by mathematical induction.

3.2 Related-Cipher Attack of Ballet-128

Assume that the 128-bit key of Ballet-128/128 is $K_{128} = k_0||k_1$, and the 256-bit key of Ballet-128/256 is $K_{256} = k_0||k_1||0||0$. Based on Property 1, we have $sk_i = rk_i$, $0 \leq i < 46$. The internal state value of Ballet-128/256 is the same as Ballet-128/128 for any plaintext. Therefore, the input state of round 47 of Ballet-128/256 can be calculated by the ciphertext of Ballet-128/128. Next, knowing the input and output values of the last two rounds of Ballet-128/256, we can recover the secret round key of these two rounds (as shown in Fig. 6) based on the round transformation.

Denote the round function of Ballet-128 as $F = \pi \circ \tau \circ \rho$, similar to Sect. 2.2, where ρ is the combination of rotation, modular addition and addition, τ is the round key addition, and π is the vector permutation of 32-bit words. The specific attack process is described as follows.

(1) Randomly choose a plaintext X, and query the corresponding ciphertexts $SY_0||SY_1||SY_2||SY_3$ and $RY_0||RY_1||RY_2||RY_3$ by calling the encryption algorithm of Ballet-128/128 and Ballet-128/256.
(2) Calculate $\rho(SY_1||SY_0||SY_3||SY_2) = Y_0||Y_1||Y_2||Y_3$, then $rk_{47}^L = Y_0 \oplus RY_1$, $rk_{47}^R = Y_3 \oplus RY_2$.
(3) Calculate $\rho^{-1}(RY_0||Y_0||Y_3||RY_3) = Z_0||Z_1||Z_2||Z_3$, then $rk_{46}^L = Z_0 \oplus SY_0$, $rk_{46}^R = Z_3 \oplus SY_3$.
(4) Due to $rk_{46} = rk_{46}^L||rk_{46}^R$ and $rk_{47} = rk_{47}^L||rk_{47}^R$, calculate the secret key $K_{128} = k_0||k_1$ by the key schedule of Ballet-128.

The time complexity of (2), (3), and (4) are both negligible. The main complexity of this attack is from one call to the encryption algorithm of Ballet-128/128 and Ballet-128/256.

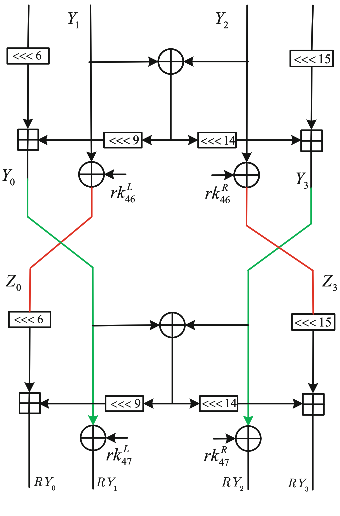


Fig. 6. Key recovery attack of Ballet-128/256

This result shows that Ballet-128 is vulnerable to related-cipher attack. It is noted that increasing the round number of Ballet-128/128 and Ballet-128/256 without changing their round number difference will not improve the ability to resist related-cipher attacks. We suggest designers should take into the total round number account in the key schedule of Ballet.

4 Evaluation of ANT Against Related-Cipher Attack

ANT-128 denotes the version with 128-bit block size and 128-bit or 256-bit key size, and their round keys are generated by feedback registers with different sizes shown in Sect. 2. Therefore, finding the right fixed round key is not very simple. Nevertheless, we still find a distinguisher relying on some restrictions on cipher keys of ANT-128/256.

Property 2. There exist a 6-round related-cipher distinguisher between ANT-128/128 and ANT-128/256, if their cipher keys satisfy $K_0 = k_0$, $K_1 = k_1$, $K_2 = sk_2$, $K_3 = sk_3$, $K_0 \oplus K_1 \oplus K_2 = 2$ and $K_1 \oplus K_2 \oplus K_3 = 6$.

Proof. The proof process consists of two parts. At first, we prove that the following equivalent relation holds.

$$sk_i = rk_i, 0 \leq i \leq 5 \tag{2}$$

From $K_0 = k_0$, $K_1 = k_1$, $K_2 = sk_2$, $K_3 = sk_3$, it is ture that $sk_0 = rk_0$, $sk_1 = rk_1$, $sk_2 = rk_2$ and $sk_3 = rk_3$. Further, combined the key schedule of ANT-128/256 and $K_0 \oplus K_1 \oplus K_2 = 2$, the 4th round key can be denoted as

$$rk_4 = rk_0 \oplus 1 \oplus rk_1 \oplus A^3(rk_3) \ = rk_2 \oplus 3 \oplus A^3(rk_3) = sk_4$$

Similarly, the following relation can be obtained with $K_1 \oplus K_2 \oplus K_3 = 6$.

$$rk_5 = rk_1 \oplus 2 \oplus rk_2 \oplus A^3(rk_4) \ = rk_3 \oplus 4 \oplus A^3(rk_4) = sk_5$$

Then, we show a 6-round related-cipher distinguisher by using the relation (2). Let P_i^{128} and P_j^{256} be the ith and jth round input of ANT-128/128 and ANT-128/256, respectively. Denote C_i^{128} and C_j^{256} as the corresponding output with round keys sk_i and rk_j. Without losing the generality, $0 \leq i \leq (5 + \Delta r)$ and $0 \leq j \leq 5$, where Δr is a small positive integer. F_j denotes the jth round function.

- Randomly choose a plaintext $X = P_0^{128} = P_0^{256}$, and query the corresponding ciphertexts $C_{5+\Delta r}^{128}$ and C_5^{256} by calling the encryption algorithm of ANT-128/128 and ANT-128/256.

It is not difficult to find the ciphertext C_5^{128} is the same as the internal state value of C_5^{256} from the context. Therefore, the $(5+\Delta r)$ round ANT-128/128 will be reduced to Δr round and the following relation holds

$$C_{5+\Delta r}^{128} = F_{5+\Delta r} \circ F_{4+\Delta r} \circ \cdots \circ F_6(C_5^{256}) \tag{3}$$

For ANT-128/128, the subkey $K_6, \cdots, K_{5+\Delta r}$ can be calculated by using Eq. (3) when Δr is enough small. In other words, the 6-round distinguisher of related-cipher works when the cost of Δr round key recovery is available.

9-Round Key Recovery Attacks. We take $\Delta r = 3$, and launch a 9-round related-cipher attack between ANT-128/128 and ANT-128/128, as shown in 7. Denote the ith round input state of ANT-128/128 as $L_i||R_i$, $0 \leq i < r$. The specific attack process is described as follows.

(1) Randomly choose a plaintext $L_0||R_0$, and query the corresponding ciphertexts $C_L||C_R$ and $RC_L||RC_R$ by calling the 9-round and 6-round encryption algorithm of ANT-128/128 and ANT-128/256, respectively. Based on Property 2, we have $L_6||R_6 = RC_L||RC_R$.

(2) Guess sk_6 and sk_8, we have

$$F(L_6) \oplus R_6 \oplus sk_6 = F(C_R) \oplus C_L \oplus sk_8 \tag{4}$$

(3) Guess sk_7 and we have

$$F(L_7) \oplus R_7 \oplus L_8 \oplus sk_7 = 0 \tag{5}$$

Since sk_6 is known, $L_7 = F(L_6) \oplus R_6 \oplus sk_6$ is determined. $R_7 = L_6$ and $L_8 = C_R$,

(4) Based on the key schedule of ANT-128/128, we have

$$sk_8 = sk_6 \oplus 7 \oplus A^3(sk_7) \tag{6}$$

(5) Joint Eq. (4) and Eq. (6) to solve for the variable sk_7, and combined with Eq. (5) to solve for the variable sk_6. Finally, obtain the variable sk_8 based on Eq. (6).

(6) Choose another plaintext $L_0^{'}||R_0^{'}$, and verify the correctness of candidates.

The time complexity of (2), (3), and (4) are both negligible. In the step (5): calculating sk_7 needs to call linear operation A and the cost is negligible, calculating sk_6 needs to call F about 2^{64} times (about $2^{60.9}$ calls to the 9-round encryption algorithm of ANT-128/128), and calculating sk_8 needs one addition operation. Therefore, the main complexity of this attack is twice calls to the 6-round encryption algorithm of ANT-128/256, and $2^{60.9}$ calls to the 9-round encryption algorithm of ANT-128/128, and we can recover 128-bit subkeys of ANT-128/128.

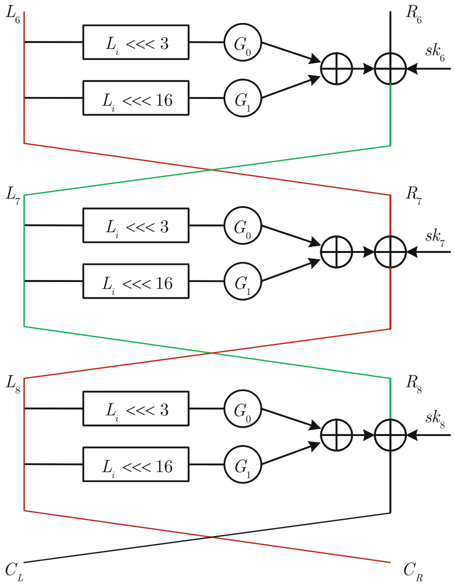


Fig. 7. Key recovery attack of ANT-128/128

Proposition 1. *There exist at most 6-round related-cipher distinguishers with equivalent keys between ANT-128/128 and ANT-128/256.*

Proof. The existence has been proved in Property 2. The maximum property can be proved by contradiction. Assume there is a 7-round related-cipher distinguisher with equivalent keys between ANT-128/128 and ANT-128/256, then there are equal round keys up to 7 rounds.

$$sk_0 = rk_0,\ sk_1 = rk_1,\ sk_2 = rk_2,\ sk_3 = rk_3,\ sk_4 = rk_4,\ sk_5 = rk_5,\ sk_6 = rk_6.$$

Based on the key schedule, the following relationships can be obtained.

$$sk_4 = sk_2 \oplus 3 \oplus A^3(sk_3),\ rk_4 = rk_0 \oplus 1 \oplus rk_1 \oplus A^3(rk_3) = sk_4 \oplus sk_2 \oplus sk_1 \oplus sk_0 \oplus 2.$$

After simplification, $sk_2 \oplus sk_1 \oplus sk_0 \oplus 2 = 0$. Similarly,

$$\begin{aligned}
&sk_5 = sk_3 \oplus 4 \oplus A^3(sk_4),\\
&rk_5 = rk_1 \oplus 2 \oplus rk_2 \oplus A^3(rk_4) = sk_5 \oplus sk_3 \oplus sk_2 \oplus sk_1 \oplus 6.\\
&sk_6 = sk_4 \oplus 5 \oplus A^3(sk_5),\\
&rk_6 = rk_2 \oplus 3 \oplus rk_3 \oplus A^3(rk_5) = sk_6 \oplus sk_4 \oplus sk_3 \oplus sk_2 \oplus 6.
\end{aligned}$$

After simplification, $sk_3 \oplus sk_2 \oplus sk_1 \oplus 6 = 0$, $sk_4 \oplus sk_3 \oplus sk_2 \oplus 6 = 0$.
In addition, the following equations hold.

$$sk_4 = sk_2 \oplus 3 \oplus A^3(sk_3),\ sk_3 = sk_1 \oplus 2 \oplus A^3(sk_2),\ sk_2 = sk_0 \oplus 1 \oplus A^3(sk_1).$$

Since the operate A is a linear function, the above three equations add up to $A^3(6) = 4$. That is, $A^3(6) = 0x180c060000000000$. Obviously, it is a contradiction. The hypothesis is wrong, and this proposition is proved.

The above result shows that the key equivalence situation can not be extended to more than 6 rounds, and ANT-128 only has low-round related-cipher attacks. Therefore, ANT-128 is relatively secure against related-cipher attack under the condition of equivalent keys.

5 Conclusion

In this paper, we summarize two basic requirements for the security of a family of ciphers against general cryptanalysis methods such as differential attack and against related-cipher cryptanalysis, and then evaluate the related-cipher attack on two block ciphers. The results are as follows. For Ballet-128/128, the full cipher key can be recovered. For Ballet-128/256, the keys of type $K_{256} = k_0||k_1||0||0$ can be recovered, and other types still need further research. For ANT-128, there are only round-reduced related-cipher attacks under the equivalent key.

The related-cipher attack applied in this paper provides a new idea for cryptanalysis work, and also helps to provide new reference criterions for the design and evaluation of new ciphers.

Acknowledgement. The authors would like to thank Dr Mir Ali Rezazadeh Baee and the anonymous reviewers for their detailed and very helpful comments and suggestions to improve this article. This work is supported by the National Natural Science Foundation of China (No. 62072445).

References

1. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN—a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_20
2. Beaulieu, R., Shors, D., Smith, J., et al.: The SIMON and SPECK lightweight block ciphers. In: Proceedings of the 52nd Annual Design Automation Conference, pp. 1–6. Association for Computing Machinery, New York (2015) . https://doi.org/10.1145/2744769.2747946
3. Yang, G., Zhu, B., Suder, V., Aagaard, M.D., Gong, G.: The Simeck family of lightweight block ciphers. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 307–329. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_16
4. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 430–454. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_17

5. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer, New York (1993). https://doi.org/10.1007/978-1-4613-9314-6
6. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_33
7. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_2
8. Knudsen, L., Wagner, D.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45661-9_9
9. Demirci, H., Selçuk, A.A.: A meet-in-the-middle attack on 8-round AES. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 116–126. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_7
10. Biham, E.: New types of cryptanalytic attacks using related keys. J. Cryptol. **7**(4), 229–246 (1994). https://doi.org/10.1007/BF00203965
11. Joan, D., Vincent, R.: The Design of Rijndael: AES: The Advanced Encryption Standard. Information Security and Cryptography, Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-662-04722-4
12. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052343
13. Ferguson, N., et al.: Improved cryptanalysis of Rijndael. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44706-7_15
14. Mala, H., Dakhilalian, M., Rijmen, V., Modarres-Hashemi, M.: Improved impossible differential cryptanalysis of 7-round AES-128. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 282–291. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17401-8_20
15. Derbez, P., Fouque, P.-A.: Exhausting Demirci-Selçuk meet-in-the-middle attacks against reduced-round AES. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 541–560. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_28
16. Wu, H.: Related-cipher attacks. In: Deng, R., Bao, F., Zhou, J., Qing, S. (eds.) ICICS 2002. LNCS, vol. 2513, pp. 447–455. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36159-6_38
17. Sung, J., Kim, J., Lee, C.: Differential related-cipher attacks on block ciphers with flexible number of rounds. Inf. Secur. Cryptol. **15**(1), 77–86 (2005)
18. Shao, Z. Y., Ding, L.: Related-cipher attack on Salsa20. In: 4th International Conference on Computational and Information Sciences on Proceedings, pp. 1182–1185. IEEE, Piscataway (2012)
19. Ding, L.: Improved related-cipher attack on Salsa20 stream cipher. IEEE Access **7**, 30197–30202 (2019)
20. Kohno, T.: Analysis of the WinZip encryption method. IACR Cryptology ePrint Archive, pp. 78(2004). https://eprint.iacr.org/2004/078
21. Cui, T.T., Wang, M.Q., et al.: Ballet: a software-friendly block cipher. J. Cryptolog. Res. **6**(6), 704–712 (2019)
22. Chen, S.Y., Fan, Y.H., Fu, Y., et al.: On the design of ANT family block ciphers. J. Cryptolog. Res. **6**(6), 748–759 (2019)

23. Notice of National Cryptgraphic Algorithm Design Competetion (in Chinese). https://www.cacrnet.org.cn/site/content/259.html. Accessed 9 Feb 2023
24. Mouha, N., Luykx, A.: Multi-key security: the even-mansour construction revisited. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 209–223. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_10
25. Wu, W.L., Zhang, L., Zheng, Y., et al.: The block cipher uBlock. J. Cryptolog. Res. **6**(6), 690–703 (2019)

Cryptanalysis of SPEEDY

Jinliang Wang[1,2], Chao Niu[1,2], Qun Liu[1,2], Muzhou Li[1,2](✉), Bart Preneel[4],
and Meiqin Wang[1,2,3]

[1] Key Laboratory of Cryptologic Technology and Information Security,
Ministry of Education, Shandong University, Jinan, China
{jinliangwang,niuchao,qunliu,muzhouli}@mail.sdu.edu.cn

[2] School of Cyber Science and Technology, Shandong University, Qingdao, China
mqwang@sdu.edu.cn

[3] Quan Cheng Shandong Laboratory, Jinan, China

[4] imec-COSIC, KU Leuven, Leuven, Belgium
Bart.Preneel@esat.kuleuven.be

Abstract. SPEEDY is a family of ultra-lightweight block ciphers designed by Leander et al. at CHES 2021. There are three recommended variants denoted as SPEEDY-r-192 with $r \in \{5, 6, 7\}$. All of them support the 192-bit block and the 192-bit key. The main focus during its design is to ensure hardware-aware low latency, thus, whether it is designed to have enough security is worth to be studied. Recently, the full-round security of SPEEDY-7-192 is announced to be broken by Boura et al. at EUROCRYPT 2023 under the chosen-ciphertext setting, where a round-reduced attack on SPEEDY-6-192 is also proposed. However, no valid attack on SPEEDY-5-192 is given due to its more restricted security parameters. Up to now, the best key recovery attack on this variant only covers 3 rounds proposed by Rohit et al. at AFRICACRYPT 2022.

In this paper, we give three full-round attacks on SPEEDY-7-192. Using the divide-and-conquer strategy and other new proposed techniques, we found a 5.5-round differential distinguisher which can be used to mount the first chosen-plaintext full-round key recovery attack. With a similar strategy, we also found a 5-round linear distinguisher which leads to the first full-round attack under the known-plaintext setting. Meanwhile, the 5.5-round differential distinguisher also helps us slightly improve the full-round attack in the chosen-ciphertext setting compared with the previous result. Besides, we also present a 4-round differential attack on SPEEDY-5-192, which is the best attack on this variant in terms of the number of rounds so far. A faster key recovery attack covering the same rounds is also given using a differential-linear distinguisher. Both attacks cannot threaten the full round security of SPEEDY-5-192.

Keywords: Lightweight Cryptography · Low Latency · SPEEDY

1 Introduction

In lightweight cryptography, cryptographic primitives are required to be suitable for resource-constrained environments, such as low area, latency or energy

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 124–156, 2023.
https://doi.org/10.1007/978-3-031-35486-1_7

consumption. Therefore, it's usually difficult to design such ciphers since one has to make trade-offs between implementation efficiency and security. Hence, the security of such ciphers is worth to be evaluated thoroughly.

In CHES 2021, Leander et al. proposed a family of ultra-low-latency block ciphers named as SPEEDY [9] to resolve the problem: How to design a secure encryption algorithm whose hardware implementation is fast. To gain such cipher, they first considered which type of logic gates and circuit topologies are suitable for ultra low-latency encryption. As a result, SPEEDY adopts a lightweight S-box, whose coordinate functions are realized as two-level NAND trees, and a hardware-friendly linear layer. After careful combination, SPEEDY achieves lower latency in hardware than most cryptographic primitives. SPEEDY contains three versions SPEEDY-5-192, SPEEDY-6-192 and SPEEDY-7-192, whose number of rounds are 5, 6, and 7, respectively. Note that, as the number of rounds decreases, the SPEEDY cipher gains better performance in hardware implementations but has a weak bound of security claim.

Recently, Boura et al. [5] announced that the full-round security of SPEEDY-7-192 is broken under the chosen-ciphertext setting, where a round-reduced attack on SPEEDY-6-192 is also proposed. However, due to the more restricted security parameters, no valid attack on SPEEDY-5-192 is given. The best key recovery attack on this variant only covers 3 rounds proposed by Rohit et al. [16].

In this paper, we aim to evaluate the security of SPEEDY thoroughly using differential cryptanalysis, linear cryptanalysis and differential-linear cryptanalysis. Proposed by Biham and Shamir in 1990, differential cryptanalysis [4] has been an important method in evaluating the security of symmetric-key ciphers. Using a differential distinguisher with high probability, one can recover the secret key after adding several rounds before or/and after it. Such an attack can be mounted only when the adversary is under the chosen-plaintext/ciphertext setting. Linear cryptanalysis [13], proposed by Matsui in 1992, only requires the adversary under the known-plaintext setting, which is more realistic than the differential attack. It exploits the linear distinguisher with high correlation considering the linear relation between some bits of plaintexts, ciphertexts and the secret key. Differential-Linear cryptanalysis [8] is another chosen plaintext/ciphertext method, which is proposed by Langford and Hellman in 1994. This method uses a distinguisher with a high correlation that consists of a differential distinguisher followed by a linear distinguisher. To evaluate its correlation more accurately, Achiya and Dunkelman introduced DLCT (short for Differential Linear Connectivity Table) [1] to connect the differential distinguisher and the linear one. This method works efficiently on ciphers whose differential probabilities and linear correlations are very high when the distinguisher covers small rounds, but decrease very quickly when the number of rounds increases.

1.1 Contributions

In this paper, we propose several key recovery attacks on full-round SPEEDY-7-192 and 4-round SPEEDY-5-192 by exploiting key-recovery friendly distinguishers constructed following the divide-and-conquer strategy. Detailed contributions are shown as follows.

Strategy of Searching Key-Recovery Friendly Differential and Linear Trail of SPEEDY. Our searching strategy includes two parts: (a) control the propagation of active pattern deduced from the input difference/mask and output difference/mask; (b) construct long distinguishers in the divide-and-conquer way. To fulfill (a), we introduce TDDT (short for truncated differential distribution table) to show the non-randomness of S-box with bit-wise truncated differential propagation. With this table, one can depict clearly in the searching algorithm on how to choose input/output differences of the target distinguisher. While for (b), we construct part of the distinguisher by concatenating two distinguishers covering the small number of rounds. For instance, the 5.5-round distinguisher is first built with a 4-round distinguisher. To gain a better 4-round distinguisher, we split it into two parts with each covering two rounds. The input difference of the MixColumn operation in the middle is chosen to minimize the total number of activated S-boxes involved in its two neighborhood S-box layers. For the other MixColumns contained in the 5.5-round distinguisher, no restriction on their input differences is necessary.

Improved Key Recovery Attacks Against Full-Round SPEEDY-7-192. By exploiting the above strategy, we gain a 5.5-round differential distinguisher, which has a higher probability than the one found by [5]. With this differential trail, we mount the first chosen-plaintext attack on its full-round variant. Besides, since it's key-recovery-friendly, it also leads to a full-round chosen-ciphertext attack, which requires slightly less complexities than the one proposed by [5]. Further, we also mount the first known-plaintext 7-round linear attack with a 5-round linear distinguisher by adopting a similar search strategy. Although this one costs slightly higher complexities than the differential attack, it is a known-plaintext attack which requires weaker capabilities of the adversary than the chosen-plaintext attack required in the differential cryptanalysis. Therefore, such linear key recovery attack is also valuable. All our attack results along with previous published ones are summarized in Table 1.

Improved Key Recovery Attack on Round-Reduced SPEEDY-5-192. We also provide two attacks on 4-round SPEEDY-5-192 using differential and differential-linear cryptanalysis, respectively. More precisely, the differential attack is mounted with a two-round distinguisher where one round is added at the top and bottom, respectively, while the differential-linear attack is given by exploiting a three-round distinguisher and adding one half-round at both sides. Note that they are the best valid attacks on it in terms of the number of rounds.

1.2 Organization

First, we briefly recall SPEEDY, as well as differential, linear and differential-linear cryptanalysis in Sect. 2. In Sect. 3, we show how to exploit the propagation properties of SB and MC operations, and introduce the TDDT (short for

Table 1. Summary of valid attacks on SPEEDY-7-192 and SPEEDY-5-192. KP, CP and CC separately denote the data collected in the known-plaintext, chosen-plaintext and chosen-ciphertext settings. Time complexities are evaluated in encryption units, while memory costs are evaluated in block size.

Version	Security Claim (Data, Time)	R	Data	Time	Memory	Type	Ref
SPEEDY-7-192	$(2^{192}, 2^{192})$	7	$2^{187.27}$CC	$2^{187.75}$	2^{42}	Diff.	[5]
		7	$2^{186.53}$CC	$2^{187.39}$	2^{36}	Diff.	Sect. 5.1
		7	$2^{186.53}$CP	$2^{187.39}$	2^{156}	Diff.	Sect. 5.1
		7	$2^{188.50}$KP	$2^{189.41}$	$2^{185.04}$	Linear	Sect. 5.2
SPEEDY-5-192	$(2^{64}, 2^{128})$	3	$2^{17.6}$CP	$2^{52.5}$	$2^{17.62}$	Integral	[16]
		4	2^{61}CC	$2^{119.69}$	2^{83}	Diff.	Sect. 5.3
		4	2^{61}CP	2^{105}	2^{105}	Diff.-Linear	Sect. 5.4

truncated differential distribution table). Using these properties, we introduce the automated search method oriented to key recovery for differential, linear and differential-linear cryptanalysis in Sect. 4. Detailed attack procedures are given in Sect. 5. Finally, we summarize our work in Sect. 6. Besides, the code for searching distinguishers in this paper is available at the following repository: https://github.com/Jin-liang-Wang/Cryptanalysis-of-SPEEDY.

2 Preliminaries

2.1 Brief Introduction of SPEEDY

SPEEDY [9] is a family of ultra-low latency block ciphers proposed at CHES 2021. SPEEDY uses a 6-bit bijective S-box and can be instantiated with different block sizes, key sizes, and numbers of rounds. For instance, SPEEDY-r-$6l$ is the version of block size and key size of $6l$-bits, and r is the number of iterated rounds. The internal state of SPEEDY-r-$6l$ can be viewed as an $l \times 6$ rectangle array of bits where $l = 32$. Following the notation of its design document [9], we use $x_{[i,j]}$ to denote the bit located at row i and column j of the state x where $0 \leq i < l, 0 \leq j < 6$.

The design document of SPEEDY suggests $l = 32$, and the number of rounds $r \in \{5, 6, 7\}$, which are called SPEEDY-5-192, SPEEDY-6-192, SPEEDY-7-192, respectively. As for the security claim, SPEEDY-r-192 achieves 128-bit security when iterated over $r = 6$ rounds and full 192-bit security when iterated over $r = 7$ rounds, while the $r = 5$ rounds variant provides 2^{128} time complexity and 2^{64} data complexity. To make it clear, we denote SPEEDY-r-192 as SPEEDY in this paper, and briefly recall the cipher as follows.

Initialization. SPEEDY receives a 192-bit plaintext and initializes the internal state with a two-dimensional matrix x. Specifically, it initializes the k-th MSB (short for the most significant bit) into $x_{[i,j]}$, where $k = 6i + j$.

Round Function. Its round function consists of five different operations: SubBox (SB), ShiftColumns (SC), MixColumns (MC), AddRoundKey (A_{k_r}) and AddRoundConstant (A_{c_r}). A high-level overview of its round function is shown in Fig. 1. Note that the last round function only has three operations.

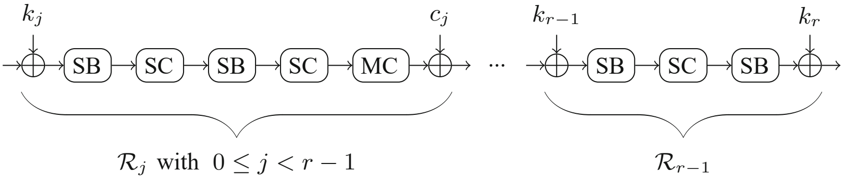


Fig. 1. r rounds of `SPEEDY` block cipher.

SB is composed of 32 parallel 6-bit Sbox, where each Sbox takes the i-th row $(x_{[i,0]}, x_{[i,1]}, x_{[i,2]}, x_{[i,3]}, x_{[i,4]}, x_{[i,5]})$ as input and its output is placed in the same row. The detail of S-box is shown in Appendix A. SC rotates the j-th column of the state upwards by j bits. In other words, the output bit $y_{[i,j]}$ equals to the input bit $x_{[(i+j) \bmod 32, j]}$. MC also operates on each column, where seven specifically chosen input bits are XORed as a new bit of its output, as shown in Eq. (1). It also can be seen as multiplying each column by a cyclic matrix.

$$y_{[i,j]} = x_{[i,j]} \oplus x_{[i+1,j]} \oplus x_{[i+5,j]} \oplus x_{[i+9,j]} \oplus x_{[i+15,j]} \oplus x_{[i+21,j]} \oplus x_{[i+26,j]}, \quad \forall i,j. \tag{1}$$

The 192-bit round key k_r and 192-bit constant c_r are respectively XORed with the entire state in A_{k_r} and A_{c_r}.

Key Schedule. The 192-bit master key of `SPEEDY` is regarded as the first round key k_0. Relation between the r-th round key k_r and the $(r+1)$-the round key k_{r+1} is constructed by a bit permutation PB. To be clear, the j-th bit of k_{r+1} equals to the i-th bit of k_r, where $j \equiv (7i+1) \bmod 192$.

2.2 Differential, Linear and Differential-Linear Crytanalysis

Differential Cryptanalysis. Differential cryptanalysis is a chosen plaintext attack proposed by Biham and Shamir [4] in 1990. Starting from a well-chosen difference δ_{in}, the distribution of $E_k(x) \oplus E_k(x \oplus \delta_{in})$ is non-uniform. More precisely, there exits a specific δ_{out} such that $\Pr_{x,k}[E_k(x) \oplus E_k(x \oplus \delta_{in}) = \delta_{out}]$ is significantly higher than 2^{-n}, where n is the block size. In order to distinguish a cipher with a high probability differential $(\delta_{in} \rightarrow \delta_{out})$ from a random permutation, one can collect D ciphertexts corresponding to pairs of plaintexts $(P, P \oplus \delta_{in})$, and compute the number of pairs following the differential:

$$Q = \#\left\{P : E_k(P) \oplus E_k(P \oplus \delta_{in}) = \delta_{out}\right\}.$$

The expected value of Q for the cipher is $D \times \Pr[\delta_{in} \rightarrow \delta_{out}]$ and $D \times 2^{-n}$ for the random permutation. Thus, the distinguisher succeeds with high probability when $D = \mathcal{O}(1/\Pr[\delta_{in} \rightarrow \delta_{out}])$.

Linear Cryptanalysis. Linear cryptanalysis, proposed by Matsui at EUROCRYPT 1993 [13], exploits the linear approximations of the round function in order to obtain a biased approximation of the cipher. The linear approximation is a pair of masks (α, α') such that the distribution of the XORed masked plaintext and ciphertext $x \cdot \alpha \oplus E_k(x) \cdot \alpha'$ is biased. More precisely, we have $(|\Pr_x[x \cdot \alpha \oplus E_k(x) \cdot \alpha'] - \frac{1}{2}| \gg 2^{-n/2})$ for most keys k), where $x \cdot y = \bigoplus_i x_i y_i$ denotes the inner product. Then, we have the correlation which is expected to be zero when averaged over all keys is:

$$\mathrm{Cor}_k\left(\alpha \rightarrow \alpha'\right) = 2\Pr_x\left[x \cdot \alpha = E_k(x) \cdot \alpha'\right] - 1.$$

Similarly, to distinguish a cipher with a biased linear approximation (α, α') from a random permutation, we collect D known plaintexts/ciphertexts, and evaluate the experimental correlation:

$$Q = \left(\#\left\{P, C : P \cdot \alpha \oplus C \cdot \alpha' = 0\right\} - \#\left\{P, C : P \cdot \alpha \oplus C \cdot \alpha' = 1\right\}\right)/D.$$

The expected value Q is larger (in absolute value) for the cipher than for a random permutation and can be observed with high probability when $D = \mathcal{O}(\mathrm{Cor}(\alpha \rightarrow \alpha')^{-2})$.

Differential-Linear Cryptanalysis. Differential-linear cryptanalysis, proposed by Langford and Hellman in 1994 [8], is a technique to combine differential and linear cryptanalysis. Let E be a cipher which can be described as a cascade of three subciphers, E_0, E_m and E_1, i.e., $E = E_1 \circ E_m \circ E_0$. Let δ_{in} and δ_{out} denote the input and output difference of E_0, α and α' be the input and output linear mask of E_1. Asume that the differential trail $\delta_{in} \rightarrow \delta_{out}$ in E_0 is satisfied with probability p, and the linear trail $\alpha \rightarrow \alpha'$ in E_1 is satisfied with correlation q. The correlation of $\delta_{out} \rightarrow \alpha$ in E_m is satisfied with

$$r = \mathrm{Cor}\left(\delta_{out} \xrightarrow{E_m} \alpha\right) = 2\Pr_x\left[E_m(x) \cdot \alpha = E_m(x \oplus \delta_{out}) \cdot \alpha\right] - 1.$$

Then, the correlation of the trail $\delta_{in} \xrightarrow{E} \alpha'$ is satisfied with correlation $\mathrm{Cor}(\delta_{in} \xrightarrow{E} \alpha') = pq^2r$. One can collect D ciphertexts corresponding to pairs of plaintexts $(P, P \oplus \delta_{in})$ and evaluate the experimental correlation:

$$Q = \left(\#\left\{P : \alpha' \cdot (E(P) \oplus E(P \oplus \delta_{in})) = 0)\right\} - \#\left\{P : \alpha' \cdot (E(P) \oplus E(P \oplus \delta_{in})) = 1)\right\}\right)/D.$$

Similarly, the expected value is larger (in absolute value) for the cipher than for a random permutation and can be observed with high probability when $D = \mathcal{O}(\mathrm{Cor}(\delta_{in} \xrightarrow{E} \alpha')^{-2})$.

3 Further Study on Operations of SPEEDY

In general, the extended path before and after the distinguisher determines the number of guessed key bits, the attack rounds, and the time complexity. To control the number of active S-boxes in the extended path, we add the constraint of the propagation through key recovery rounds and propose the concept of TDDT (short for truncated differential distribution table). Since the security of SPEEDY highly relies on the 6-bit S-box and the MC operation, we did in-depth research for SB operation with the TDDT in Sect. 3.1 and for MC operation in Sect. 3.2, respectively. With these newly observed properties, we can construct the automated search method for the differential and linear distinguishers by taking the key recovery into account in Sect. 4.

3.1 Truncated Differential Distribution Table

In general, the time complexity of differential cryptanalysis is affected by the number of active bits in plaintext and ciphertext. To control the number of active bits, we limit some bits being inactive before the second SB operation to reduce the number of active bits in plaintext, as shown in Sect. 4.

On average, each bit is limited inactive in rounds of key recovery with probability 2^{-1}. However, we have found that in a specific S-box, e.g., the S-box of SPEEDY, this probability is fluctuate considerably. Two examples are provided in Equations (2a) and (2b) below, although the input sets of these two situations are both one bit inactive, the probabilities of Eqs. (2a) and (2b) are much higher than 2^{-1}.

$$\texttt{**0***} \xrightarrow[p=2^{-0.54}]{\textsf{SB}} \texttt{010000}\,, \tag{2a}$$

$$\texttt{0*****} \xrightarrow[p=2^{-0.83}]{\textsf{SB}} \texttt{010000}\,, \tag{2b}$$

where * denotes a arbitrary bit $\in \{0,1\}$. The probability is computed as follows. Taking Eq. (2b) as an example. The input set Δ_{in} contains 2^5 differences, i.e., $\{\texttt{0b000000},\texttt{0b000001},\cdots,\texttt{0b011111}\}$. We assume that each difference in Δ_{in} appears with equal probability. Besides, the output difference is $\delta_{out} = \texttt{0b010000}$. Then this probability is computed by $\Pr(\Delta_{in} \xrightarrow{\textsf{SB}} \delta_{out}) = \frac{\sum_{\delta_{in}\in\Delta_{in}} \Pr(\delta_{in} \xrightarrow{\textsf{SB}} \delta_{out})}{|\Delta_{in}|}$.

In order to get an optimal probability in rounds of key recovery, we need to consider how to select inactive bits. In this section, we introduce the TDDT and show how it can select inactive bits using the automatic solver in Sect. 4.

Compared to DDT (short for differential distribution table [4]) which contains the number of transition situations of each fixed input-output difference, for a N-bit S-box, the TDDT is a $2^N \times 2^N$ table which contains the number of transition situation from a set of input differences to a set of output differences. To clarify, we define the following notation to represent a set of differences named bit-string $\mathbf{k}$. For an n-bit bit-string $\mathbf{k} = (k_0, k_1, ..., k_{n-1}), \mathbf{k} \in \mathbb{F}_2^n$, we define $\mathbf{k}' \preceq \mathbf{k}$, if $k_i' \leq k_i$ for all i. Then, we denote $\mathcal{B}_{\Delta_{in}}/\mathcal{B}_{\Delta_{out}}$ as the bit-string representation of the input/output difference set Δ_{in}/Δ_{out}. For example, when a truncated input difference Δ_{in} is "00*00*", the bit-string representation $\mathcal{B}_{\Delta_{in}}$ is $\texttt{0b001001}_{(\mathcal{S})}$[1],

[1] To distinguish the set of differences from the single differential, we use numbers with subscript $_{(\mathcal{S})}$ to represent the set of differences.

i.e., $\mathcal{B}_{\Delta_{in}} = \texttt{0b001001}_{(\mathcal{S})}$ denotes the differential set "00*00*" which contains differences {0b000000,0b000001,0b001000,0b001001}. Thus, we can construct TDDT from the DDT using Algorithm 1 and the probability of the truncated difference propagation from Δ_{in} to Δ_{out} is:

$$\Pr(\Delta_{in} \xrightarrow{\mathsf{SB}} \Delta_{out}) = \frac{\mathsf{TDDT}[\mathcal{B}_{\Delta_{in}}][\mathcal{B}_{\Delta_{out}}]}{|\Delta_{in}| \cdot 2^N}, \tag{3}$$

where N is the size of S-box and $|\Delta_{in}|$ denotes the number of input differences δ_{in} that fulfills $\delta_{in} \preceq \Delta_{in}$.

```
Algorithm 1: Construct TDDT
Input: S-box
Output: TDDT; // Truncated difference distribution table
$\mathcal{L}[i][j] \leftarrow 0,(0 \le i, j < 2^N)$; // Initial an empty list for TDDT
$\mathcal{L}[0][0] \leftarrow 2^N$; // Initialize the case $0 \xrightarrow{\mathsf{SB}} 0$ with probability 1
Generate the DDT of S-box;
for $\mathcal{B}_{\Delta_{in}} \in [1_{(\mathcal{S})}, 2^N_{(\mathcal{S})})$ do
    for $\mathcal{B}_{\Delta_{out}} \in [1_{(\mathcal{S})}, 2^N_{(\mathcal{S})})$ do
        for $\delta_{out} \preceq \Delta_{out}$ do
            for $\delta_{in} \preceq \Delta_{in}$ do
                $\mathcal{L}[\mathcal{B}_{\Delta_{in}}][\mathcal{B}_{\Delta_{out}}] \leftarrow \mathcal{L}[\mathcal{B}_{\Delta_{in}}][\mathcal{B}_{\Delta_{out}}] + \mathsf{DDT}[\delta_{in}][\delta_{out}]$;
return $\mathcal{L}$ as TDDT of the S-box;
```

By exploiting the differential property of the transition from a set of differences that are inactive in certain bits to another set of differences with different inactive bits, TDDT can help us make a universal search model for key recovery. Moreover, we also consider the cases of differential propagation from the fixed input difference to a set of output differences. Then, we define FTDDT(short for fixed-input truncated difference distribution table) to describe the probability distribution of this transition and introduce it as follows.

Fixed-Input Truncated Difference Distribution Table. As aforementioned, FTDDT contains the number of transition situations of a fixed input difference to a set of output differences. For the special case, when the input difference is zero, the output difference must be zero, i.e., the set of output differences Δ_{out} contains only zero differences. More precise, when $\delta_{in} = 0$, we have

$$\mathsf{FTDDT}[0][0] = 2^N \quad \text{and} \quad \mathsf{FTDDT}[0][i] = 0, i \in [1, 2^N).$$

We show the construction of FTDDT in Algorithm 2. By modeling these difference distribution tables into our automated search model, we can find a key recovery-oriented differential which contains key recovery rounds as well as distinguishers. Time complexity of Algorithm 2 is$(2^N - 1) \cdot \sum_{i=1}^{N} \binom{N}{i} 2^i = \mathcal{O}(2^{N \cdot \log_2 6})$, which is evaluated as the number of the look-up table operations.

Algorithm 2: Construct FTDDT

```
Input: S-box;
Output: FTDDT; // Fixed input/output truncated difference
    distribution table
$\mathcal{L}[i][j] \leftarrow 0$, $(0 \leq i, j < 2^N)$; // Initialize an empty list for FTDDT
$\mathcal{L}[0][0] \leftarrow 2^N$; // Initial the case $0 \xrightarrow{\mathsf{SB}} 0$ with probability 1
Generate the DDT of S-box;
for $\delta_{in} \in [1, 2^N)$ do
    for $\Delta_{out} \in [1_{(\mathcal{S})}, 2^N_{(\mathcal{S})})$ do
        for $\delta_{out} \preceq \Delta_{out}$ do
            $\mathcal{L}[\delta_{in}][\Delta_{out}] \leftarrow \mathcal{L}[\delta_{in}][\Delta_{out}] + \mathsf{DDT}[\delta_{in}][\delta_{out}]$;
return $\mathcal{L}$ as FTDDT of the S-box;
```

In summary, FTDDT shows the non-randomness of S-box with bit-wise truncated differential propagation. Considering the differential propagation through a random permutation, the fixed input difference δ_{in} will propagate to the output difference set Δ_{out} with probability 2^{-i} where the number of zero bits in $\mathcal{B}_{\Delta_{out}}$ is i. As mentioned before, due to the non-randomness of a specific S-box, the probability is changed to $\mathsf{FTDDT}[\delta_{in}][\mathcal{B}_{\Delta_{out}}]/2^N$. In order to verify the correctness of FTDDT, we have conducted an experiment in Appendix B. With this accurate propagation probability of differential, we can gain a better-extended path for a key recovery attack as shown in Sect. 4.

Inverse Probability of FTDDT. During key recovery, we need to partially encrypt (resp. decrypt) the plaintext pair (resp. ciphertext pair) to validate the input (resp. output) of the distinguisher. To filter the good pairs of plaintext more precisely in validating the input of the distinguisher, we utilize the inverse of FTDDT. We denote N_a as the number of active bits in Δ_{out}. For example, $N_a = 5$ if $\Delta_{out} = \texttt{0b110111}$, i.e., **0***. Then, when considering a fixed difference $\delta_{out} \preceq \Delta_{out}$ propagating to δ_{in} which is the inverse of $\mathsf{FTDDT}[\delta_{in}][\mathcal{B}_{\Delta_{out}}]$, the average probability of this transition is $\mathsf{FTDDT}[\delta_{in}][\mathcal{B}_{\Delta_{out}}]/2^{N+N_a}$. The proof is shown in Appendix C. With the inverse probability of FTDDT, we will get the more precise time complexity of the key-recovery procedure.

3.2 Differential Propagation Property of MC

The MC operation of SPEEDY generates each new bit in a column by XORing seven current bits. It also can be seen as multiplying each column by a cyclic matrix M. In this section, we research the MC operation in-depth by considering the matrix M. Here we traverse all possible inputs with HW (short for Hamming weight) from one to eight and find that their minimum output HW are $(7, 8, 7, 8, 7, 6, 5, 4)$. Table 2 summarizes which choices of active differential bits are associated with these minimal output HW cases. Specifically, we use $\{a_0, a_1, \cdots\}$ to denote that the a_i-th bit of the input difference is active. Moreover, due to the property of the MC operation of SPEEDY, we rotate the array and make the 0-th bit active to represent the rotation invariant array class. From Table 2, we can observe that when the input HW is one, there is a minimum sum of the input and output minimum HW. Meanwhile, when the input HW is two and three, the minimal sum of the input and output HW is relatively small.

Table 2. Summary of active differentials yielding low output HW from M. When the input HW is greater than 3, the cases of active bits are too large to list. We record their number to simplify. For the input differences with different HW, we list its output difference which have the minimum HW.

In HW	Out HW	Active Bits	Total HW
1	7	{0}	8
2	8	{0, 6}	10
		{0, 11}	10
3	7	{0, 4, 15}	10
4	8	10 cases	12
5	7	7 cases	12
6	6	8 cases	12
7	5	8 cases	12
8	4	5 cases	12

When building the automated search model, one can utilize the property of MC to control the diffusion of the differential, which can reduce the size of the search space. In this work, we utilize a divide-and-conquer method to search for a longer differential by dividing it into several parts. More precisely, for a four rounds differential, we first search several two rounds differentials with the input difference of the last MC is of relatively low HW as in Table 2. Then, we search the last two rounds of differential and connect them with the middle MC. By limiting the **sum** of input-output HW of the middle MC, we can control the diffusion of the 4-round differential, *i.e.* minimizing the total number of activated S-boxes involved in the middle, and achieve a higher probability.

We note here that Boura et al. [5] proposed a different strategy to utilize the properties of the MC operation. In their approach, a differential is divided into several one-round parts and they consider the case that the HW of **both** input and output are small.

4 Automated Search Method Oriented to Key Recovery

In this section, we introduce the automated search strategies of differential, linear and differential-linear distinguishers against SPEEDY following the divide-and-conquer strategy, as well as taking the key recovery procedure into consideration.

Usually, cryptanalysts perform key recovery attacks based on a strong distinguisher. By appending several rounds before and after the distinguisher, one can obtain an extended path containing both distinguisher and key recovery rounds. However, this two-step process of key recovery cannot promise optimal results as shown in [15,18]. Therefore, automated search oriented to key recovery can obtain better attack results. To find such key-recovery-friendly distinguishers, properties of SB and MC are introduced in Sect. 3 are utilized.

We implement our search algorithms using STP[2], which is a solver developed based on the SAT (short for boolean satisfiability problem) [12]/SMT(short for satisfiability modulo theories) [2] problem and has been widely used in the field of cryptography [6,10,11,14].

4.1 Differential Search Model Against SPEEDY

Since we aim to search for distinguishers in the single-key setting, we can omit A_{k_r} and A_{c_r} in our model. We denote S_j^i as the j-th intermediate state of the i-th round and set the first round indexed with zero as the key recovery round.

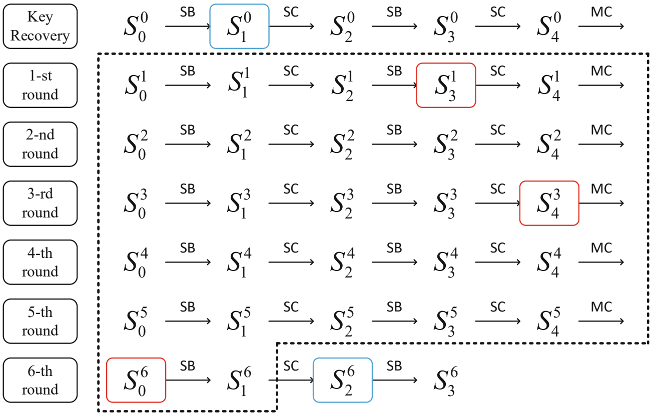

Key Recovery
1-st round
2-nd round
3-rd round
4-th round
5-th round
6-th round
SB
SC
MC
S_0^0 S_1^0 S_2^0 S_3^0 S_4^0
S_0^1 S_1^1 S_2^1 S_3^1 S_4^1
S_0^2 S_1^2 S_2^2 S_3^2 S_4^2
S_0^3 S_1^3 S_2^3 S_3^3 S_4^3
S_0^4 S_1^4 S_2^4 S_3^4 S_4^4
S_0^5 S_1^5 S_2^5 S_3^5 S_4^5
S_0^6 S_1^6 S_2^6 S_3^6

Fig. 2. 7-round differential attack of SPEEDY.

Generally, the variable number and the size of each variable will determine the total scale of the search model and affect the complexity of solving the SAT/SMT problem. As the number of rounds increases, the time and memory

[2] https://stp.github.io/.

complexity of solving the problem increases exponentially. Specifically, when the model exceeds four rounds, we cannot obtain a solution in a reasonable time with STP in the case of SPEEDY. For obtaining a longer distinguisher, we employ a divide-and-conquer [18] approach to search several models with a short round to form the entire differential. As shown in Fig. 2, our automated search model is divided into several steps. Motivated by [15], we also consider a key recovery-oriented search model where the distinguisher rounds and key recovery rounds are all included.

Model One: 7-Round Differential Contains 1.5-Round of Key Recovery. The 7-round differential consists of 5.5-round differential distinguisher $S_0^1 \rightarrow S_1^6$ and 1.5 key recovery rounds $S_0^0 \rightarrow S_0^1, S_1^6 \rightarrow S_3^6$. For convenience, we use backward FTDDT to represent the FTDDT of SPEEDY S-box and forward FTDDT to represent the FTDDT of the inverse of SPEEDY S-box.

Step 1: From S_3^1 to S_4^3, the search strategies are as follows.

Rule 1: Model SB to comply with DDT.

Rule 2: Model each active S-box in S_3^1 to comply with $2^{-3\times 2}$ probability. This strategy is mainly used to limit the active S-box in the S_3^1. More precisely, we measure this trail with probability $2^{-6\times N_a} \times p$, where N_a is the number of active S-box in S_3^1 and p is the probability generated from **Rule** 1 in **Step** 1. Indeed, N_a active S-boxes imply that there will have $2 \times N_a$ S-boxes in trail $S_0^1 \rightarrow S_3^1$. Since the maximum probability transition through the S-box is 2^{-3}, we measure each active S-box in S_3^1 with probability 2^{-6}.

Rule 3: Constrain only one of the columns to be activated in S_4^3. For the input before MC, consider only entries with low HW from Table 2.

Step 2: From S_4^3 to S_0^6, the strategies to reduce search space are as follows.

Rule 1: Due to the differential must be linked, we fix the input differences as those in S_4^3.

Rule 2: Model the active S-box in S_0^6 to comply with a maximum probability of 2^{-3}.

Step 3: After running the above two steps in parallel, we obtain several four rounds of differentials with high probability. Then, we choose the differential whose probability is higher than 2^{-170} and search the corresponding last 1.5-round differential of distinguisher and 1.5-round of key recovery as follows.

Rule 1: Model $S_0^1 \xrightarrow{\mathsf{SB}} S_1^1$ and $S_2^1 \xrightarrow{\mathsf{SB}} S_3^1$ to comply with DDT.

Rule 2: Model $S_1^0 \xrightarrow{\mathsf{SB}^{-1}} S_0^0$ and $S_0^6 \xrightarrow{\mathsf{SB}} S_1^6$ to comply with FTDDT.

Rule 3: Constrain the active S-box in S_1^0 and S_2^6 less than 32.

Rule 4: Constrain the active S-box number in S_1^0 less than 16.

Rule 5: Constrain the active S-box number in S_2^6 more than 6.

Here, rules 3 to 5 are used to constrain the active bits to reduce the time complexity of the key recovery phase as shown in Sect. 5.1.

Step 4: Following step one to three, we can get several 5.5-round distinguishers with 1.5-round of key recovery. Then, we choose the distinguisher with maximum probability and search an extended path for key recovery as shown in Algorithm 6 (Appendix G). To reduce the time complexity, we consider the differential rotation invariant of SPEEDY. More precisely, we use one of the 32 rotation-invariant distinguishers to implement the key recovery attack, and the chosen one makes the lowest time complexity.

With the above four steps, we find a differential trail with probability $2^{-185.53}$ which can be used to attack on the 7-round SPEEDY. More precisely, this trail consists of a 5.5-round differential distinguisher with probability $2^{-182.49}$ and a 1.5-round key-recovery phase with probability $2^{-3.04}$. We illustrate it as Fig. 3.

Model Two: 4-Round Differential Contains Two Rounds of Key Recovery. From **Step 2**, we can get several two rounds distinguisher from S_4^3 to S_0^6. Then, we use backward FTDDT and forward FTDDT to search 2-round of key recovery. We show this distinguisher with probability $2^{-59.35}$ in Appendix J. For the property of linear key relation between k_0 and k_4, the rotation invariant property does not affect the time complexity when attacking the 4-round SPEEDY.

4.2 Searching Linear Distinguishers Against SPEEDY

For linear cryptanalysis, we use a similar strategy used for differential search, which is a model oriented toward key recovery. Then, we search for a five-round distinguisher with two rounds of key recovery by following a four-step process. Due to the page number limit of the paper, we have placed the specifics in Appendix D.

Finally, we found a 5-round distinguisher combined with two key recovery rounds with correlation $2^{-93.01}$. We show the detail of the linear distinguisher in Appendix I. In this distinguisher, the bits of k_7 which is derived from k_0 by linear key schedule is 139. We reduce the guessed key bits to 185 bits in the key recovery phase.

4.3 Searching Differential-Linear Distinguishers Against SPEEDY

Similarly, we search for distinguishers against 4-round SPEEDY. More specifically, we search for three rounds of differential-linear distinguishers and extend half a round forward and backward for key recovery. The detailed strategy is shown in Appendix E.

Finally, the distinguisher is shown in Fig. 5 of Appendix F. The correlation of this 3-round distinguisher is $2^{-23.095}$ and the probability of rounds of key-recovery is $2^{-6.972}$. The bits of k_7 deduced from k_0 by linear key schedule are 29.

5 Key Recovery Attack Against SPEEDY

In this section, we first implemented both differential and linear cryptanalysis on **full** SPEEDY with the newly obtained distinguishers in Sect. 4. Then, the key recovery attack against other versions of SPEEDY are also provided.

5.1 7-Round Differential Attack Against Full SPEEDY

Using the newly obtained 7-round differential containing distinguishers and key recovery rounds, we can perform key recovery attacks on **full** SPEEDY. Before the attack, we first introduce some techniques that will be used in this section as follows.

Deduce k_0 from k_7 by Linear Key Schedule. Due to the linear key schedule, we can get the relationship between k_0 and k_7 and deduce the key bits of k_0 from k_7. We call these bits of k_0 deduced key bits from k_7. The detailed relationship between k_0 and k_7 is shown as the purple squares in Fig. 3.

Function to Sieve Possible Key Guesses. The FirstSboxSieve function uses the key bits of k_7 to deduce the six bits of each row in k_0 using the linear key schedule. After deducing the six bits of k_0 from k_7, we can filter the corresponding key guessing of the plaintext pair with the zero difference in S_1^0. Then, for each plaintext pair and the corresponding six bit keys of each row, there will leave 2^{6-d-n} key guesses after sieving by the first S-box, where d is the bit number deduced from k_7 and n is the inactive bit number after the first S-box. We show the details of FirstSBoxSieve in Algorithm 3. The SecondSboxSieve is used to combine the sets L_i of possible key bits in a bigger set. The function can sieve the elements of the bigger set using the difference in S_3^0. The detail is shown in Algorithm 4.

Distinguisher with Rounds of Key Recovery. The 5.5-round differential distinguisher is shown in Fig. 3, where the black square and the blue square denote the active bits from S_4^1 to S_4^5 and the active bits from S_3^0 to S_3^1, respectively. To distinguish the differential propagation of the key recovery round, we use different colors to indicate the propagation of each active bit in the extended rounds, where the gray square indicates the active bits whose differences are forced zero. Further, we use the shaded square to denote the bits random from $\{0, 1\}$. Then, we use the red square to denote the active bits in S_0^0 and S_3^6. To represent the bits in k_0 that are derived from k_7 using the linear key schedule, we use the purple square to denote the deduced bits in k_0 and the green square to denote the guessed key bits when performing the key recovery attack. Note that 180 key bits in k_0 are involved in the key recovery attack.

The attack procedure contains three parts, i.e., data collection, key recovery and brute force the master key.

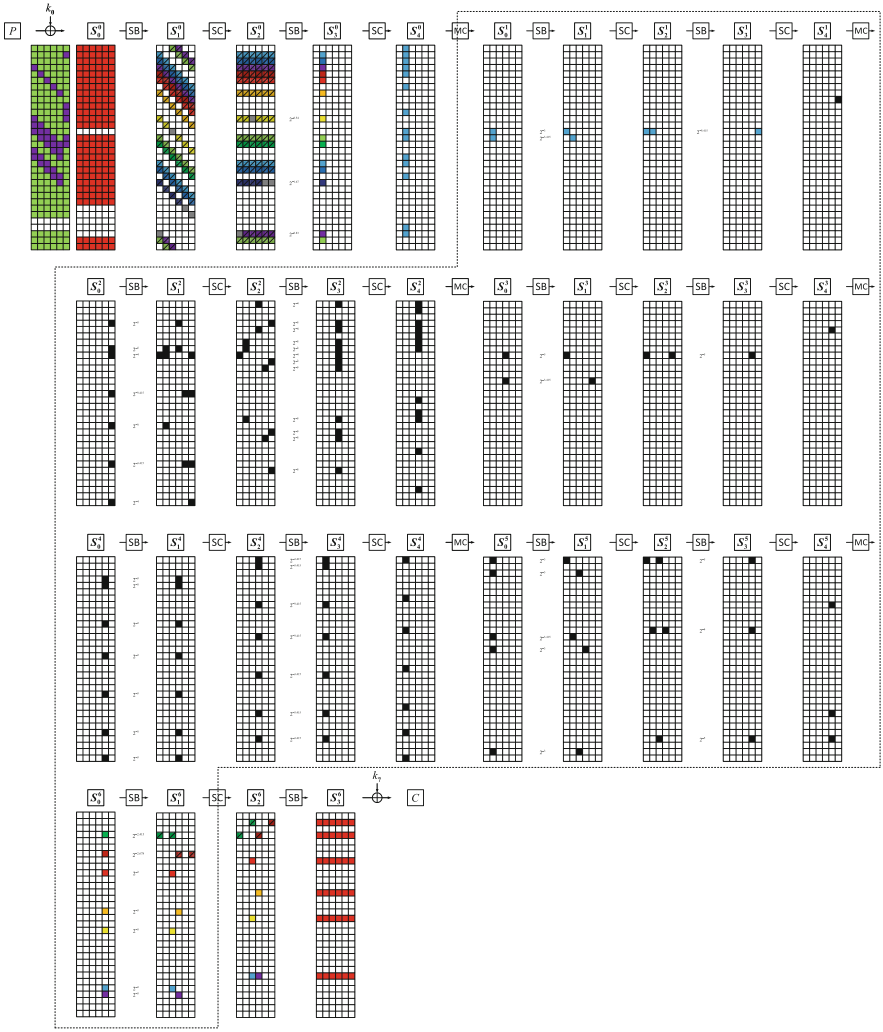


Fig. 3. Key recovery attack on 7-round SPEEDY with differential cryptanalysis. When the output set of a FTDDT has only one single bit, it is shown as a single difference in this figure since it has one possible case for such an output set, e.g., 0b00*000 is shown as 0b001000 since the active difference is impossible to transit to the 0b000000 through the S-box of SPEEDY.

Algorithm 3: Sieving keys with the first S-box

```
Input: $(m_0^i, m_1^i)$; // The $i$-th row of the plaintext pair
Input: $\Delta_k$; // The set of 6-bit keys
Input: $\delta_{out}$; // The output difference for right pair
Output: Possible 6-bit keys for the $i$-th row with the corresponding
        compressed internal state of pairs in $S_1^0$;
function FirstSboxSieve($m_0^i, m_1^i, \Delta_k, \delta_{out}$)
    $\mathcal{L} \leftarrow \emptyset$;
    for all $k_i \in \Delta_k$ do
        $x_0 \leftarrow$ S-box($m_0 \oplus k_i$);
        $x_1 \leftarrow$ S-box($m_1 \oplus k_i$);
        if $x_0 \oplus x_1 == \delta_{out}$ then
            Compress $x_0$ by the active bits in $\delta_{out}$;
            Compress $x_1$ by the active bits in $\delta_{out}$;
            $\mathcal{L} \overset{\cup}{\leftarrow} (x_0, x_1, k_i)$;
    if $\mathcal{L} == \emptyset$ then
        Discard the 6-bit last round key for this $(m_0^i, m_i^j)$ pair;
    else
        return $\mathcal{L}$;
```

Data Collection. Consider structure S which consists of 2^{156} plaintexts and for each plaintext pair $P_0, P_1 \in S$, $P_0 \oplus P_1 = (N,N,N,N,N,N,N,N,N,N,N, N,N, 0,N,N,N,N,N,N,N,N,N,N,N,0,0,0,0,0,N,N)$, where N denotes any 6-bit cell. We require the oracle to encrypt those 2^{156} plaintexts and collect the ciphertexts by the *hash* table which is indexed by the 156-bit inactive bits in ciphertexts. From the *hash* table, we expect to get 2^{155} pairs of plaintext-ciphertext pair $(P_0, C_0), (P_1, C_1)$ that satisfy $C_0 \oplus C_1 = (0,N,0,N,0,0,0,N,0,0,0,0,N,0,0, 0,N,0,0,0,0,0,0,0,0,N,0,0,0,0,0,0)$.

Next, we briefly introduce the key recovery and brute force phases, the detail of the pseudocode is shown in Appendix G. We use δ_{in}^i to denote the i-th row of the input difference of the 5.5-round distinguisher.

Key Recovery. For each pair obtained from the data collection phase, we first use $C_0 \oplus C_1$ to deduce the 36-bit k_7 from DDT. Note that, since we evaluate the propagation probability of key-recovery rounds by FTDDT, the number of deduced 36-bit k_7 should be calculated by the inverse probability of FTDDT as shown in Sect. 3.1. For each deduced 36-bit k_7, we fix the deduced keys in k_0 from k_7 due to the linear key schedule and guess other 144-bit k_0 as follows.

Firstly, we use the early abort technique which is guessing the key-bits of row 14,15,17 and partial encrypt to run FisrtSboxSieve. If there are no satisfied key guesses to validate the output difference of the SubBox operation, we consider the 36-bit deduced k_7 is wrong and discard it. If all the

Algorithm 4: Sieving keys with the second S-box

```
Input: $[\mathcal{L}_i, ..., \mathcal{L}_{i+n-1}]$; // The sets obtained from FisrtSboxSieve or
    SecondSboxSieve
Input: $\delta_{out}$; // The output difference for right pair in $S_3^0$
Output: The set of possible keys with the corresponding compressed internal
        state of pairs in $S_3^0$.
function SecondSboxSieve($m_0^i, m_1^i, \Delta_k, \delta_{out}$)
    $\mathcal{L} \leftarrow \emptyset$;
    for all $(x_0^i, x_1^i, k_i) \in \mathcal{L}_i$ do
        for all $(x_0^{i+n-1}, x_1^{i+n-1}, k_{i+n-1}) \in \mathcal{L}_i$ do
            $x_0 \leftarrow$ ComActBit($x_0^i, \cdots x_0^{i+n-1}$);
            $x_1 \leftarrow$ ComActBit($x_1^i, \cdots x_1^{i+n-1}$);
            /* ComActBit function is used to combine the bits
               propagated by the $\delta_{out}$ in $(x_t^i, \cdots x_t^{i+n-1})$ into $x_t$       */
            $y_0 \leftarrow$ S-box($x_0$);
            $y_1 \leftarrow$ S-box($x_1$);
            if $y_0 \oplus y_1 == \delta_{out}$ then
                $key \leftarrow k_i||...||k_{i+n-1}$;
                $x_0 \leftarrow$ SaveBit($x_0^i, \cdots x_0^{i+n-1}$);
                $x_1 \leftarrow$ SaveBit($x_1^i, \cdots x_1^{i+n-1}$);
                /* SaveBit function is used to save the bits used in
                   the following algorithm                                */
                $\mathcal{L} \overset{\cup}{\leftarrow} (x_0, x_1, key)$;
    if $\mathcal{L} == \emptyset$ then
        Discard the 6-bit last round key for this $(m_0^i, m_i^j)$ pair;
    else
        return $\mathcal{L}$;
```

deduced k_7 are wrong, we consider this pair of plaintext-ciphertext is wrong and discard it. Next, for each plaintext-ciphertext pair, we sieve k_0 in row `11,12,13,16` with FisrtSboxSieve and partial encrypt to δ_{in}^{11} with SecondSboxSieve. After this step, the plaintext-ciphertext pair and corresponding k_0 can validate the input difference δ_{in}^{11}. Similarly, we discard the deduced k_7 if there are no satisfied k_0 guesses to validate δ_{in}^{11} and discard the pair if no deduced k_7 is left. Then, we employed the same method to those deduced keys in the row of k_0 with FisrtSboxSieve and partial encrypt plaintext pairs to $\delta_{in}^{14}, \delta_{in}^{15}, \delta_{in}^{18}, \delta_{in}^{19}, \delta_{in}^{7}, \delta_{in}^{5}, \delta_{in}^{4}, \delta_{in}^{3}, \delta_{in}^{2}, \delta_{in}^{1}, \delta_{in}^{30}, \delta_{in}^{29}, \delta_{in}^{21}$ to further sieve with SecondSboxSieve.

Finally, it will leave $2^{135.46}$ pairs and $2^{6.5}$ 180-bit keys for each pair on average, and we will get $2^{141.96}$ candidate keys of this 180-bit master key for each structure. Then, we use the brute force procedure to sieve the wrong candidate keys.

Brute Force. For each candidate key from the key recovery procedure, we guess the remaining 12-bit of the master key to get the complete 192-bit candidate master key k^c. Then we use a test pair (m_t, c_t) from encrypt oracle to check if $E_{k^c}(m_t) = c_t$ to verify if k^c is the right key.

Complexity and Success Probability. Since the memory access of storing the ciphertexts to the *hash* table is negligible compared to the time for collecting the ciphertexts, the time complexity of data collection for each structure is 2^{156}. Since the *hash* table storing 2^{155} pairs is very large, we treat the time to access this table as one encryption. Further, since the 7-round SPEEDY encryption has $7 \times 2 \times 32 = 448$ SubBox operations, the time complexity of the key recovery phase is less than $2^{155} + (70 \times 2^{155})/448$ for each structure (details are shown in Appendix H). Then, the time complexity of Brute Force phase is $2^{151.96}$ times 7-round SPEEDY encryption for each structure. Overall, the total time complexity of each structure is $2^{156} + 2^{155} + (70 \times 2^{155})/448 + 2^{153.96} \approx 2^{156.86}$ 7-round SPEEDY encryptions.

The signal-to-noise-ratio is Using $2^{185.53}/2^{155} \approx 2^{30.53}$ structures, we can ensure the success rate of 79.15% with 7.51-bit advantage according to [17]. Hence, the data complexity is $2^{186.53}$, the time complexity is $2^{30.53} \times 2^{156.86} \approx 2^{187.39}$ and the memory required is 2^{156}.

Reducing Memory Requirement. The above attack require 2^{156} bits in memory, which is too large. To obtain lower memory complexity, we can use the chosen ciphertext attack mode and select the active 36-bit structure at the ciphertext. After using the inactive 36 bits in the plaintext as the index, we get 2^{35} pairs, and then one can follow the key recovery procedure used in the above attack to obtain the right key.

5.2 7-Round Linear Attack Against Full SPEEDY

With the 5-round linear distinguisher shown in Appendix I, we compute the attack parameter on SPEEDY-7-192 using the symbols borrowed from [7]. With the $\mathcal{FWT}$ approach in [7], the time complexity is $2(\rho_M + \rho_A)2^{|k_0|+|k_7|-l_{07}}$, where the $|k_i|$ means the bit number guessed in k_i, and l_{07} means the key bits of k_7 deduced from k_0. Following the notation in [7], ρ_A is the cost of an addition, and ρ_M is the cost of a multiplication. Assuming that two multiplications correspond to roughly one evaluation of the cipher, we have the time complexity of key recovery phase 2^{186} encryptions. The memory required is $2^{144} + 2^{180} + 2^{185} \approx 2^{185.04}$. According to [17], we need $2^{188.50}$ data and the advantage $a = 4$ to achieve the success rate 69.12%. Finally, we exhaustively search the remaining 7 key bits. The total time complexity of the attack is $2^{188.50} + 2^{186} + 2^{188} \approx 2^{189.41}$.

5.3 4-Round Differential Attack Against SPEEDY.

As shown in Appendix J, we choose ciphertexts in S_3^3 to make a 60-bit structure and filter the corresponding plaintext pairs in S_0^2. After running this step, it will leave $\binom{2^{60}}{2}/2^{36} \approx 2^{83}$ pairs. Then, we sieve the key bits of the 108-bit involved k_4 according to the 2^{83} pairs and leave $2^{43.692}$ possible k_4 for each pair. Then, for the combination of these $2^{83} \times 2^{43.692} = 2^{126.692}$ plaintext-ciphertext pairs and corresponding key guesses, we guess the 156-bits k_0 using the method in Sect. 5.1.

Due to the differential probability being $2^{-59.35}$ and one structure can only provide 2^{60} data, we construct two structures to collect the data. The data complexity of this attack is 2^{61}, the time complexity is $2^{119.692}$ and the memory required is 2^{83}. With above attack parameters, the success rate is 94.17%.

5.4 4-Round Differential-Linear Attack Against SPEEDY

With the distinguisher searching in Sect. 4.3, we obtain a 4-round differential-linear attack against SPEEDY with 2^{61} data complexity 2^{105} time complexity and 2^{105} memory requirement. The detail of this attack is shown in Appendix F.

6 Conclusion

In this paper, we first show how to find key-recovery friendly distinguishers for SPEEDY by following the divide-and-conquer strategy as well as some other new techniques. With such strategies, we found a 5.5-round differential distinguisher which is key-recovery friendly and with higher probability than the one used in the previous result. Using this distinguisher, we are able to mount the first chosen-plaintext full-round attack on SPEEDY-7-192. Besides, with the same distinguisher, we can also mount a full-round attack under the chosen-ciphertext setting, which slightly reduces the attack complexities compared with the previous one proposed in the same setting. Meanwhile, using a similar search strategy, we also found a 5-round linear distinguisher which leads to the first known-plaintext full-round attack. Although the full-round security of this variant has recently been announced to be broken, our attacks here need a weaker ability requirement for the adversary. Besides, for SPEEDY-5-192, which requires more restricted attack parameters, we also propose two 4-round key recovery attacks using a differential distinguisher and a differential-linear one, respectively. According to our best knowledge, both attacks are the best ones on this variant in terms of the number of rounds. However, its full-round security is not threatened.

Acknowledgements. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This work is supported by the National Key Research and Development Program of China (Grant No. 2018YFA0704702), the National Natural Science Foundation of China (Grant No. 62032014), the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025). The corresponding author is also supported by Qingdao Innovation project (Grant No. QDBSH20230101008).

A The 6-Bit S-Box of SPEEDY

(See Table 3).

Table 3. The 6-bit S-box of SPEEDY. All elements are expressed in hexadecimal.

x_0x_1	$x_2x_3x_4x_5$															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	08	00	09	03	38	10	29	13	0c	0d	04	07	30	01	20	23
1	1a	12	18	32	3e	16	2c	36	1c	1d	14	37	34	05	24	27
2	02	06	0b	0f	33	17	21	15	0a	1b	0e	1f	31	11	25	35
3	22	26	2a	2e	3a	1e	28	3c	2b	3b	2f	3f	39	19	2d	3d

B Experiment to Verify the Correctness of FTDDT

In this section, we conducted an experiment to calculate the probability of propagation to verify the correctness of FTDDT. For several random-chosen fixed-input differences δ_{in}, we tested the transfer probability from a single difference δ_{in} (shown in the left-hand side of Eq. 4) to a set of differences Δ_{out} (shown in the right-hand side of Eq. 4) in the S-box used in SPEEDY.

More precisely, we randomly choose 2^{20} 42-bit plaintexts p and denote the number of p that satisfies $\{\text{S-box}(p) \oplus \text{S-box}(p \oplus \delta_{in}) \in \Delta_{out}\}$ as $\mathcal{N}$. Then the probability obtained from this test is $\mathcal{N}/2^{20}$.

$$\begin{gathered}
\mathsf{010000} \xrightarrow{\mathsf{SB}} \mathsf{{*}{*}0{*}{*}{*}} \\
\mathsf{010000} \xrightarrow{\mathsf{SB}} \mathsf{{*}{*}{*}{*}00} \\
\mathsf{010000} \xrightarrow{\mathsf{SB}} \mathsf{0{*}{*}{*}{*}{*}} \\
\mathsf{001000} \xrightarrow{\mathsf{SB}} \mathsf{00{*}{*}{*}{*}} \\
\mathsf{001000} \xrightarrow{\mathsf{SB}} \mathsf{{*}{*}{*}{*}00} \\
\mathsf{000100} \xrightarrow{\mathsf{SB}} \mathsf{{*}{*}{*}0{*}{*}} \\
\mathsf{000100} \xrightarrow{\mathsf{SB}} \mathsf{0{*}{*}0{*}{*}}
\end{gathered} \tag{4}$$

Our validation algorithm is written in python code and we run it on an Intel(R) Core(TM) i7-8700 CPU. Finally, the tested probability is $2^{-5.9671}$ while the transfer probability caculated from FTDDT is $2^{-5.9622}$. Therefore, we consider that the accuracy obtained by FTDDT is verified.

C Inverse Probability of FTDDT

Theorem 1. When considering a fixed difference $\delta_{out} \preceq \Delta_{out}$ propagate to δ_{in} which is the inverse of $\mathsf{FTDDT}[\delta_{in}][\mathcal{B}_{\Delta_{out}}]$, the average probability of this transition is

$$\mathsf{FTDDT}[\delta_{in}][\mathcal{B}_{\Delta_{out}}]/2^{N+N_a}.$$

Proof. Since the S-box is bijective, the inverse probability of FTDDT is equal to the average probability of $\delta_{in} \xrightarrow{\mathsf{SB}} \delta_{out}$ (averaged over all $\delta_{out} \in \Delta_{out}$). Next, remember that the FTDDT describes the probability of a group of differential transition. Thus, with this probability, we can obtain the average probability of $\delta_{in} \xrightarrow{\mathsf{SB}} \delta_{out}, \delta_{out} \preceq \Delta_{out}$ by dividing by the size of the output differential set $|\Delta_{out}|$. The probability of the case $\delta_{in} \xrightarrow{\mathsf{SB}} \Delta_{out}$ is $\mathsf{FTDDT}[\delta_{in}][\mathcal{B}_{\Delta_{out}}]/2^N$ and the number of difference in Δ_{out} is 2^{N_a}. Thus, the probability of the case $\delta_{in} \xrightarrow{\mathsf{SB}} \delta_{out}$ is $\mathsf{FTDDT}[\delta_{in}][\mathcal{B}_{\Delta_{out}}]/2^{n+N_a}$ on average if $\delta_{out} \in \Delta_{out}$. □

D Searching Linear Distinguishers Against SPEEDY

Since the MC operation can be seen as $\Gamma_{in} = \mathsf{M}^{\mathsf{T}} \cdot \Gamma_{out}$ in linear attack where Γ_{in} is the input mask and Γ_{out} is the output mask of matrix M. Following the method in Sect. 3.2, we get the linear mask transition properties for MC, precisely, the correspondence between the input mask Γ_{in} and the output mask Γ_{out} of the minimum HW.

For linear cryptanalysis, we use a similar strategy used for differential search, which is a model oriented toward key recovery. Then, we search for a five-round distinguisher with two rounds of key recovery by following four steps.

Step 1: We found the 4-round distinguishers from S_3^0 to S_0^5 using the same method in searching differentials.

Step 2: For each of the 4-round linear distinguishers, we found the distinguishers with two rounds of key recovery from S_0^5 to S_2^6 according to the following conditions: the probability from S_0^5 to S_0^6 is as high as possible and after spreading from S_0^6 to S_2^6 with probability one, the active S-box number in S_2^6 is less than 32.

Step 3: Then, the linear mask propagates from S_3^0 to S_1^0 with probability one and limits the number of active S-box at S_1^0 is less than 32.

Step 4: Finally, we consider the linear rotation invariant of the 7-round distinguishers combined with key recovery and choose the case where the number of bits of k_7 deduced from k_0 by linear key schedule is maximized.

Following step one to four, we found a 5-round distinguisher combined two key recovery rounds with correlation $2^{-93.01}$. We show the detail of the linear distinguisher in Appendix I. In this distinguisher, the bits of k_7 which is derived from k_0 by linear key schedule is 139. We reduce the guessed key bits to 185 bits in the key recovery phase.

E Searching Differential-Linear Distinguishers Against SPEEDY

We show the four rounds differential-linear search model in Fig. 4. More specifically, we search for three rounds of differential-linear distinguishers from S_4^0 to S_0^4 and extend half a round forward and backward for key recovery. Detailed strategy is as follows.

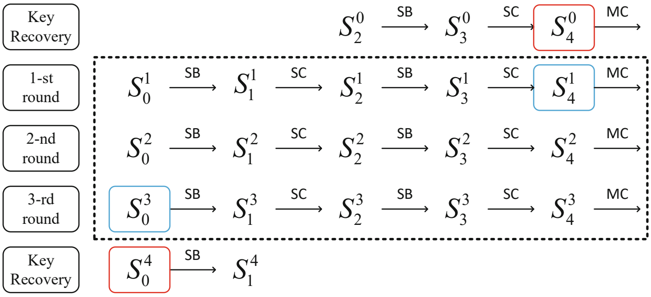

Key Recovery
1-st round
2-nd round
3-rd round
Key Recovery
S_2^0 SB S_3^0 SC S_4^0 MC
S_0^1 SB S_1^1 SC S_2^1 SB S_3^1 SC S_4^1 MC
S_0^2 SB S_1^2 SC S_2^2 SB S_3^2 SC S_4^2 MC
S_0^3 SB S_1^3 SC S_2^3 SB S_3^3 SC S_4^3 MC
S_0^4 SB S_1^4

Fig. 4. 4-round differential-linear attack model of SPEEDY.

Step 1: We search the differential part contains key recovery rounds from S_3^0 to S_4^1, the search strategies are as follows.

Rule 1: Constrain the active bit numbers in S_3^0 less than 64.

Rule 2: Model $S_3^0 \xrightarrow{\mathsf{SB}^{-1}} S_2^0$ to comply with FTDDT.

Rule 3: Model $S_0^1 \xrightarrow{\mathsf{SB}} S_1^1, S_2^1 \xrightarrow{\mathsf{SB}} S_3^1$ to comply with DDT.

Rule 4: Constrain only one bit active in S_4^1.

Step 2: Meanwhile, we search the linear part contains key recovery rounds from S_0^3 to S_0^4, the search strategies are as follows.

Rule 1: Constrain only one bit active in S_0^3.

Rule 2: Model $S_0^3 \xrightarrow{\mathsf{SB}} S_1^3, S_2^3 \xrightarrow{\mathsf{SB}} S_3^3$ to comply with LAT (short for linear approximation table).

Rule 2: Model the active S-box in S_0^4 to comply with a maximum correlation of $2^{-1.415}$.

Step 3: We search the connect round from S_4^1 to S_0^3 as follows.

Rule 1: Constrain only one bit active in S_4^1 and S_0^3.

Rule 2: Model $S_0^2 \xrightarrow{\mathsf{SB}} S_1^2$ with backward FTDDT.

Rule 3: Model $S_2^2 \xrightarrow{\mathsf{SB}} S_3^2$ to comply with LAT.

Rule 4: Constrain the difference of the i-th row in S_2^2 is zero if the difference of corresponding row in S_2^2 is active.

Step 4: From Step one to three, we get several distinguishers. Then, we choose the one with the highest correlation and run an experiment for obtaining the estimated correlation from S_2^1 to S_1^3. After testing 128 random keys with 2^{30} plaintexts under a certain key, we obtained the correlation estimated as $2^{-8.85}$.

Step 5: Finally, we consider the rotation invariant property of SPEEDY such that the bits of k_7 deduced from k_0 by linear key schedule are maximized.

Finally, the distinguisher is shown in Fig. 5. The correlation of this 3-round distinguisher is $2^{-23.095}$ and the probability of rounds of key-recovery is $2^{-6.972}$. The bits of k_7 deduced from k_0 by linear key schedule are 29.

F 4-Round Differential-Linear Attack Against SPEEDY

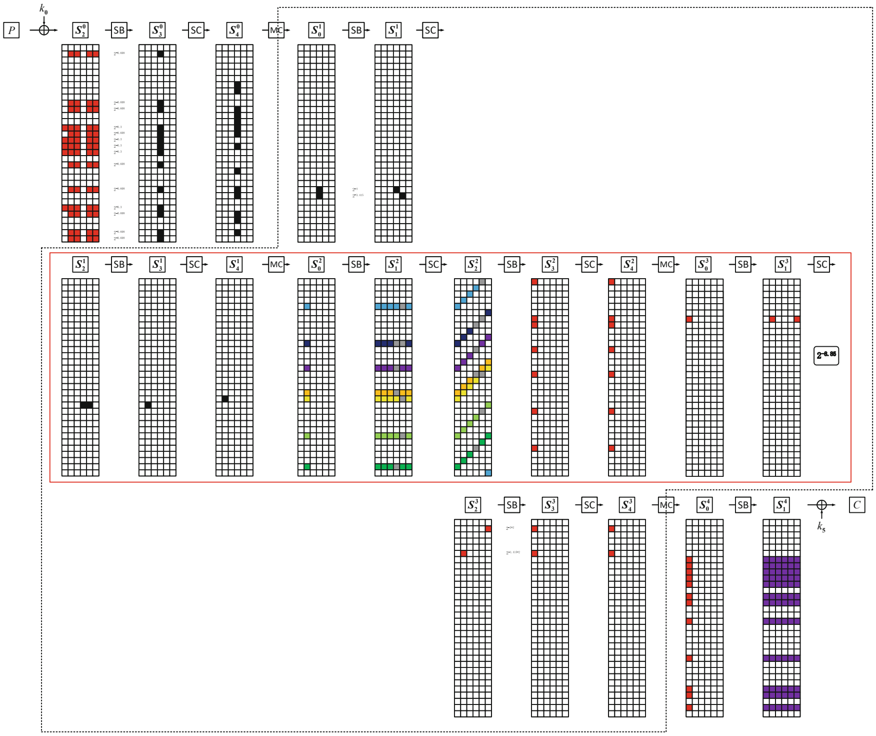


Fig. 5. Key recovery attack on 4-round SPEEDY with differential-linear cryptanalysis.

With the 3-round differential-linear distinguisher shown in Fig. 5, we can implement the differential-linear attack against 4-round SPEEDY. Firstly, we choose the active bits in plaintext to make a 61-bit structure and obtain corresponding ciphertexts. After guessing the 61-bit k_0, we use Algorithm 5 to put the pairs that can validate the input difference of the 3-round differential-linear distinguisher into the set $\mathcal{S}$. Then we recover the master key by applying the FWHT (short for Fast Walsh-Hadamard Transform) which has already been used in [3].

Since the time complexity of the Algorithm 5 is negligible compared to those of FWHT, we use the same computing method for our attack. Then, we choose the advantage $a = 88$, use one structure to collect data, the time complexity of this attack is $2^{61} + 2^{104} + 2^{104} \approx 2^{105}$. The data complexity is 2^{61} and the memory required is $2^{61} + 2^{72} + 2^{105} \approx 2^{105}$. We denote $|\mathcal{S}|$ as the number of pairs in $\mathcal{S}$, according to [17] the distribution of $|\mathcal{S}|$ is

$$|\mathcal{S}| \sim \frac{1}{2} N(2^{61} \cdot 2^{-6.972}, 2^{61} \cdot 2^{-6.972}) = N(2^{53.028}, 2^{52.028}).$$

Then, the success rate is

$$\begin{aligned} P_S &= \int_0^{+\infty} P(|\mathcal{S}| = x) \cdot P_{S'}(|\mathcal{S}| = x) dx \\ &\geq P(|\mathcal{S}| \geq 2^{53.027}) \cdot P_{S'}(|\mathcal{S}| = 2^{53.027}) \\ &\approx 45.31\%, \end{aligned}$$

where $P_{S'}(|\mathcal{S}| = n)$ is the success rate for this attack, if $|\mathcal{S}| = n$.

Algorithm 5: Sieving pairs for differential-linear attack

```
Input: $\mathcal{P}$; // The set of plaintexts
Input: $\mathcal{C}$; // The set of ciphertexts indexed by plaintexts
Input: $k$; // The guessed 61-bit $k_0$
Input: $\delta_{in}$; // The begin of differential-linear distinguisher
Output: $\mathcal{S}$; // The set of pairs can encrypt to the begin of
    differential-linear distinguisher by $k$
$\mathcal{S} \leftarrow \emptyset$;
for $p \in \mathcal{P}$ do
    $p' \leftarrow D(E(p \oplus k) \oplus \delta_{in}) \oplus k$ ;
    /* $E(\cdot)$ is the function encrypt plaintext to the begin of
        distinguisher                                              */
    /* $D(\cdot)$ is the decrypt function of $E(\cdot)$            */
    $c \leftarrow \mathcal{C}[p]$;
    $c' \leftarrow \mathcal{C}[p']$;
    $\mathcal{S} \overset{\cup}{\leftarrow} ((p, c), (p', c'))$;
Delete duplicate pairs in $\mathcal{S}$;
return $\mathcal{S}$;
```

G Pseudocode of Key Recovery and Brute Force Parts on the 2^{155} Pairs for Each Structure in the 7-Round Differential Attack

Algorithm 6: Deduce key from the data set $\mathcal{P}$

```
Input: $\mathcal{P}$; // The set of $2^{155}$ pairs, each pair consist of two
    plaintexts with their ciphertexts
Input: $\delta_{out}$; // The output difference of the 5.5-round
    distinguisher
Input: $\delta_{in}^{i}$; // The input difference of the $i$-th row of
    5.5-round distinguisher
Input: $(m_t, c_t)$; // $c_t$ is the ciphertext of $m_t$. At the end of
    this algorithm, we use it to test if the candidate keys
    are the right key.
Output: $\mathcal{K}$; // The set of possible full master key
$\mathcal{K} \leftarrow \emptyset$;
for $((m_0, c_0), (m_1, c_1)) \in \mathcal{P}$ do
    $m_0^0||...||m_0^{31} \leftarrow$ DividePlain$(m_0)$;
    $m_1^0||...||m_1^{31} \leftarrow$ DividePlain$(m_1)$;
    /* DividePlain$(m_i)$ is use to divide plaintext $m_i$ into 32
        row, $m_i^j$ is the 6-bit $j$-th row of $m_i$ */
    $\mathcal{K}^7 \leftarrow$ Deduce the 36-bit $k_7$ by $m_0 \oplus m_1$ and $\delta_{out}$ according to DDT;
        // $|\mathcal{K}^7| = 4$ on average for each pair, details are shown
        in Appendix H.
    for $k^7 \in \mathcal{K}^7$ do
        Deduce the $k_0$ from $k^7$ due to the linear key schedule;
        For $i$-th row of $k_0$, fix the deduced bits and traverse other bits to
         get set $\mathcal{K}_i^0, i = 0, \cdots, 26, 29, 30, 31$;
        $\mathcal{L}_{14} \leftarrow$ FirstSboxSieve$(m_0^{14}, m_1^{14}, \mathcal{K}_{14}^0$,0b*00*00);
        $\mathcal{L}_{15} \leftarrow$ FirstSboxSieve$(m_0^{15}, m_1^{15}, \mathcal{K}_{15}^0$,0b**00*0);
        $\mathcal{L}_{17} \leftarrow$ FirstSboxSieve$(m_0^{17}, m_1^{17}, \mathcal{K}_{17}^0$,0b00**00);
        $\mathcal{L}_{11} \leftarrow$ FirstSboxSieve$(m_0^{11}, m_1^{11}, \mathcal{K}_{11}^0$,0b*000*0);
        $\mathcal{L}_{12} \leftarrow$ FirstSboxSieve$(m_0^{12}, m_1^{12}, \mathcal{K}_{12}^0$,0b0*000*);
        $\mathcal{L}_{13} \leftarrow$ FirstSboxSieve$(m_0^{13}, m_1^{13}, \mathcal{K}_{13}^0$,0b******);
        $\mathcal{L}_{16} \leftarrow$ FirstSboxSieve$(m_0^{16}, m_1^{16}, \mathcal{K}_{16}^0$,0b0**00*);
        $\mathcal{L} \leftarrow$ SecondSboxSieve$([\mathcal{L}_{11}, \mathcal{L}_{12}, \mathcal{L}_{13}, \mathcal{L}_{14}, \mathcal{L}_{15}, \mathcal{L}_{16}], \delta_{in}^{11})$;
        $\mathcal{L}_{18} \leftarrow$ FirstSboxSieve$(m_0^{18}, m_1^{18}, \mathcal{K}_{18}^0$,0b*00**0);
        $\mathcal{L}_{19} \leftarrow$ FirstSboxSieve$(m_0^{19}, m_1^{19}, \mathcal{K}_{19}^0$,0b**000*);
        $\mathcal{L} \leftarrow$ SecondSboxSieve$([\mathcal{L}, \mathcal{L}_{17}, \mathcal{L}_{18}, \mathcal{L}_{19}], \delta_{in}^{14})$;
        $\mathcal{L}_{20} \leftarrow$ FirstSboxSieve$(m_0^{20}, m_1^{20}, \mathcal{K}_{20}^0$,0b0**00*);
        $\mathcal{L} \leftarrow$ SecondSboxSieve$([\mathcal{L}, \mathcal{L}_{20}], \delta_{in}^{15})$;
        $\mathcal{L}_{21} \leftarrow \textsc{FirstSboxSieve}(m_0^{21}, m_1^{21}, \mathcal{K}_{21}^0, \texttt{0b*0**00});$
        $\mathcal{L}_{22} \leftarrow \textsc{FirstSboxSieve}(m_0^{22}, m_1^{22}, \mathcal{K}_{22}^0, \texttt{0b0*0**0});$
        $\mathcal{L}_{23} \leftarrow \textsc{FirstSboxSieve}(m_0^{23}, m_1^{23}, \mathcal{K}_{23}^0, \texttt{0b00*0**});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}, \mathcal{L}_{21}, \mathcal{L}_{22}, \mathcal{L}_{23}], \delta_{in}^{18});$
        $\mathcal{L}_{24} \leftarrow \textsc{FirstSboxSieve}(m_0^{24}, m_1^{24}, \mathcal{K}_{24}^0, \texttt{0b000*0*});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}, \mathcal{L}_{24}], \delta_{in}^{19});$
        $\mathcal{L}_{7} \leftarrow \textsc{FirstSboxSieve}(m_0^{7}, m_1^{7}, \mathcal{K}_{7}^0, \texttt{0b*0****});$
        $\mathcal{L}_{8} \leftarrow \textsc{FirstSboxSieve}(m_0^{8}, m_1^{8}, \mathcal{K}_{8}^0, \texttt{0b0*0***});$
        $\mathcal{L}_{9} \leftarrow \textsc{FirstSboxSieve}(m_0^{9}, m_1^{9}, \mathcal{K}_{9}^0, \texttt{0b00*0**});$
        $\mathcal{L}_{10} \leftarrow \textsc{FirstSboxSieve}(m_0^{10}, m_1^{10}, \mathcal{K}_{10}^0, \texttt{0b000*0*});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([L_7, L_8, L_9, L_{10}, \mathcal{L}], \delta_{in}^{7});$
        $\mathcal{L}_{5} \leftarrow \textsc{FirstSboxSieve}(m_0^{5}, m_1^{5}, \mathcal{K}_{5}^0, \texttt{0b*****0});$
        $\mathcal{L}_{6} \leftarrow \textsc{FirstSboxSieve}(m_0^{6}, m_1^{6}, \mathcal{K}_{6}^0, \texttt{0b0*****});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}_5, \mathcal{L}_6, \mathcal{L}], \delta_{in}^{5});$
        $\mathcal{L}_{4} \leftarrow \textsc{FirstSboxSieve}(m_0^{4}, m_1^{4}, \mathcal{K}_{4}^0, \texttt{0b****00});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}_4, \mathcal{L}], \delta_{in}^{4});$
        $\mathcal{L}_{3} \leftarrow \textsc{FirstSboxSieve}(m_0^{3}, m_1^{3}, \mathcal{K}_{3}^0, \texttt{0b***00*});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}_3, \mathcal{L}], \delta_{in}^{3});$
        $\mathcal{L}_{2} \leftarrow \textsc{FirstSboxSieve}(m_0^{2}, m_1^{2}, \mathcal{K}_{2}^0, \texttt{0b**00**});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}_2, \mathcal{L}], \delta_{in}^{2});$
        $\mathcal{L}_{1} \leftarrow \textsc{FirstSboxSieve}(m_0^{1}, m_1^{1}, \mathcal{K}_{1}^0, \texttt{0b*00**0});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}_1, \mathcal{L}], \delta_{in}^{1});$
        $\mathcal{L}_{30} \leftarrow \textsc{FirstSboxSieve}(m_0^{30}, m_1^{30}, \mathcal{K}_{30}^0, \texttt{0b**0000});$
        $\mathcal{L}_{31} \leftarrow \textsc{FirstSboxSieve}(m_0^{31}, m_1^{31}, \mathcal{K}_{31}^0, \texttt{0b0**000});$
        $\mathcal{L}_{0} \leftarrow \textsc{FirstSboxSieve}(m_0^{0}, m_1^{0}, \mathcal{K}_{0}^0, \texttt{0b00**00});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}_{30}, \mathcal{L}_{31}, \mathcal{L}_0, \mathcal{L}], \delta_{in}^{30});$
        $\mathcal{L}_{29} \leftarrow \textsc{FirstSboxSieve}(m_0^{29}, m_1^{29}, \mathcal{K}_{29}^0, \texttt{0b******});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}([\mathcal{L}_{29}, \mathcal{L}], \delta_{in}^{29});$
        $\mathcal{L}_{25} \leftarrow \textsc{FirstSboxSieve}(m_0^{25}, m_1^{25}, \mathcal{K}_{25}^0, \texttt{0b******});$
        $\mathcal{L}_{26} \leftarrow \textsc{FirstSboxSieve}(m_0^{26}, m_1^{26}, \mathcal{K}_{26}^0, \texttt{0b******});$
        $\mathcal{L} \leftarrow \textsc{SecondSboxSieve}(\mathcal{L}, \mathcal{L}_{25}, \mathcal{L}_{26}], \mathcal{L}_{21});$
    **for** $k' \in \mathcal{L}$ **do**
        **for** $k''$ *traverse unguessed 12-bit keys* **do**
            Combine $k'$ and $k''$ to get the 192-bit keys $k$;
            **if** $E(m_t, k) == c_t$ **then**
                $\mathcal{K} \xleftarrow{\cup} k;$
**return** $\mathcal{K}$ as the set of candidate keys;
```

H Details for the Time Complexity of the Key Recovery on the 2^{155} Pairs for Each Structure in the 7-Round Differential Attack

In this section, we mainly consider the time of operating the Algorithm 6 in Appendix G. In order to make it easier to understand, we show the details of each step in Algorithm 6 in Table 4.

Position in Table 4 denotes the location in Algorithm 6, e.g., position 9 denotes the operation that generates $\mathcal{L}_{14}$ at line 9 in Algorithm 6. The filtering probability of FirstSboxSieve is determined by the number of inactive bits in the output of the S-box, e.g., when generating the $\mathcal{L}_{14}$, the output different set of the S-box is `0b*00*00` so that the filtering probability is 2^{-4} while the filtering probability of SecondSboxSieve is calculated by the inverse probability of FTDDT.

Besides, to facilitate the calculation of the time complexity, we see the 2^{155} pair of each structure as a group. Then, we compute the time complexity of each operation in Algorithm 6 in this group. More precisely, we introduce two new parameters, i.e., the number of the remaining pairs and the number of remaining keys for each pair. Further, for an operation with filtering probability p, the number of the remaining pairs and remaining keys before this operation is N_p and N_k, respectively. Besides, we use N_a to denote the number of imported keys in this operation. For the FisrtSboxSieve operation, I_k is the number of keys in the input key set, e.g., $|\mathcal{K}^0_{14}|$ in position 9. For the SecondSboxSieve operation, I_a is zero. Then, the complexity of this operation for the whole group in this operation, the remaining pairs and the remaining keys after this operation is calculated by Algorithm 7.

In fact, the sum of the time complexity in Table 4 is about 68.8×2^{155} times SubBox operation. Since there are other operations that have not been evaluated, we give a generous estimate for the complexity of the key recovery phase, i.e., 70×2^{155} times SubBox operation.

Evaluation the Number of Elements in $\mathcal{K}^7$. In this section, we use $S^i_j[k,:]$ to denote the k-th line in the S^i_j, i.e., the $6k$-th bit to $(6k+5)$-th bit of S^i_j. As shown in Fig. 3, there are only two active differences transitioning to a set of differences through the process $S^6_0 \xrightarrow{\mathsf{SB}} S^6_1$, i.e., the $S^6_0[3,:] \xrightarrow{\mathsf{SB}} S^6_1[3,:]$ (`0b000010`$\xrightarrow{\mathsf{SB}}$ `0b*0*000`) and $S^6_0[6,:] \xrightarrow{\mathsf{SB}} S^6_1[6,:]$ (`0b000010` $\xrightarrow{\mathsf{SB}}$`0b000*0*`). Other active differences propagate to a single difference.

Therefore, when deducing the key bits of k_7 by DDT (position 5 in Algorithm 6), on average, there is only one 6-bit partial key will be deduced for the states $k_7[i,:]$, where $i \in \{7, 2, 16, 25\}$ since the corresponding difference in S^6_2 is a specific single difference. However, $k_7[1,:]$ and $k_7[3,:]$ will have several possibilities since the corresponding state $S^6_2[1,:]$ and $S^6_2[3,:]$ are sets of differences. Thus, the average number of elements in $\mathcal{K}^7$ is determined by the average number of deduced $k_7[1,:]$ and $k_7[3,:]$.

Algorithm 7: Compute parameters in Table 4.

```
Input: $I_k$: the number of imported keys in this operation.
Input: $N_p$: the number of remaining pairs before this operation.
Input: $N_k$: the number of remaining keys before this operation.
Input: $p$: the number of remaining keys before this operation.
Output: $T$: time complexity of this operation.
Output: $N_p$: the number of remaining pairs after this operation.
Output: $N_k$: the number of remaining keys after this operation.
if This operation is FirstSboxSieve then
    $T \leftarrow 2 \times I_k \times N_p$;
    // Each pair has two plaintexts so that need two SubBox operation
       for an imported key.
    if $I_k \times p \geq 1$ then
        $N_k \leftarrow N_k \times I_k \times p$;
        $N_p \leftarrow N_p$;
    if $I_k \times p < 1$ then
        $N_k \leftarrow N_k$;
        $N_p \leftarrow N_p \times I_k \times p$;
if This operation is SecondSboxSieve then
    $T \leftarrow 2 \times N_k \times N_p$;
    if $N_k \times p \geq 1$ then
        $N_k \leftarrow N_k \times p$;
        $N_p \leftarrow N_p$;
    if $N_k \times p < 1$ then
        $N_k \leftarrow 1$;
        $N_p \leftarrow N_p \times N_k \times p$;
return $T, N_p, N_k$;
```

Note that the state $S_2^6[1,:]$ and $S_2^6[3,:]$ are propagated by $S_1^6[3,:]$ and $S_1^6[6,:]$ while the state $S_1^6[i,:]$ is propagated by $S_0^6[i,:]$ through the S-box. Thus, we did an in-depth study of the propagation $S_0^6[i,:] \xrightarrow{\mathsf{SB}} S_0^6[i,:]$, where $i \in \{3, 6\}$.

- In our trail, the $S_0^6[3,:] \xrightarrow{\mathsf{SB}} S_0^6[3,:]$ will have two cases since the propagation `0b000010` $\xrightarrow{\mathsf{SB}}$ `0b*0*000` have two possible propagations, i.e., `0b000010` $\xrightarrow{\mathsf{SB}}$ `0b100000` and `0b000010` $\xrightarrow{\mathsf{SB}}$ `0b001000` (indeed, there are four cases for this propagation, but other two cases have zero probability).
- For the same reason, the $S_0^6[6,:] \xrightarrow{\mathsf{SB}} S_0^6[6,:]$ also has two cases.

Above all, there are $2 \times 2 = 4$ possible cases for the propagation of the two active S-boxes and each case will deduce one possible 12-bit keys for k_7. Besides, each case needs two Subox operations. Thus, the average number of elements in $\mathcal{K}^7$ is and the time complexity of deducing 36-bit k_7 is $4 \times 2 + 4 = 12$ times SubBox operation.

Table 4. Details for the Time Complexity of Algorithm 6. The set parameter is the output of the operation, e.g., $\mathcal{L}_{14}$ for position 9. The time complexity is measured by the SubBox operation. I_k is the number of imported keys in this operation. N_p and N_k are the number of the remaining pairs and remaining keys after the operation calculated by Algorithm 7.

Position	set	I_k	Time complexity	Filtering probability	N_p	N_k
5	$\mathcal{K}^7$	–	12×2^{155}	1	2^{155}	4
9	$\mathcal{L}_{14}$	2^2	2^{158}	2^{-4}	2^{155}	1
10	$\mathcal{L}_{15}$	2^2	2^{158}	2^{-3}	2^{154}	1
11	$\mathcal{L}_{17}$	2^3	2^{158}	2^{-4}	2^{153}	1
12	$\mathcal{L}_{11}$	2^4	2^{158}	2^{-4}	2^{153}	1
13	$\mathcal{L}_{12}$	2^4	2^{158}	2^{-4}	2^{153}	1
14	$\mathcal{L}_{13}$	2^3	2^{157}	1	2^{153}	2^3
15	$\mathcal{L}_{16}$	2^3	2^{157}	2^{-3}	2^{153}	2^3
16	$\mathcal{L}$	0	2^{157}	$2^{-5.54}$	$2^{150.46}$	1
17	$\mathcal{L}_{18}$	2^4	$2^{155.46}$	2^{-3}	$2^{150.46}$	2
18	$\mathcal{L}_{19}$	2^4	$2^{155.46}$	2^{-2}	$2^{150.46}$	2^3
19	$\mathcal{L}$	0	$2^{154.46}$	2^{-6}	$2^{147.46}$	1
20	$\mathcal{L}_{20}$	2^4	$2^{152.46}$	2^{-3}	$2^{147.46}$	2
21	$\mathcal{L}$	0	$2^{149.46}$	2^{-6}	$2^{142.46}$	1
24	$\mathcal{L}_{21}$	2^5	$2^{148.46}$	2^{-3}	$2^{142.46}$	2^2
25	$\mathcal{L}_{22}$	2^6	$2^{149.46}$	2^{-3}	$2^{142.46}$	2^5
26	$\mathcal{L}_{23}$	2^6	$2^{149.46}$	2^{-3}	$2^{142.46}$	2^8
27	$\mathcal{L}$	0	$2^{151.46}$	2^{-6}	$2^{142.46}$	2^2
28	$\mathcal{L}_{24}$	2^6	$2^{149.46}$	2^{-4}	$2^{142.46}$	2^4
29	$\mathcal{L}$	0	$2^{147.46}$	2^{-6}	$2^{140.46}$	1
30	$\mathcal{L}_7$	2^5	$2^{146.46}$	2^{-1}	$2^{140.46}$	2^4
31	$\mathcal{L}_8$	2^6	$2^{147.46}$	2^{-2}	$2^{140.46}$	2^8
32	$\mathcal{L}_9$	2^5	$2^{146.46}$	2^{-3}	$2^{140.46}$	2^{10}

(*continued*)

Table 4. (*continued*)

Position	set	I_k	Time complexity	Filtering probability	N_p	N_k
33	$\mathcal{L}_{10}$	2^5	$2^{146.46}$	2^{-4}	$2^{140.46}$	2^{11}
34	$\mathcal{L}$	0	$2^{152.46}$	2^{-6}	$2^{140.46}$	2^5
35	$\mathcal{L}_5$	2^5	$2^{146.46}$	2^{-1}	$2^{140.46}$	2^9
36	$\mathcal{L}_6$	2^5	$2^{146.46}$	2^{-1}	$2^{140.46}$	2^{13}
37	$\mathcal{L}$	0	$2^{154.46}$	2^{-6}	$2^{140.46}$	2^7
38	$\mathcal{L}_4$	2^5	$2^{146.46}$	2^{-2}	$2^{140.46}$	2^{10}
39	$\mathcal{L}$	0	$2^{151.46}$	2^{-6}	$2^{140.46}$	2^4
40	$\mathcal{L}_3$	2^5	$2^{146.46}$	2^{-2}	$2^{140.46}$	2^7
41	$\mathcal{L}$	0	$2^{148.46}$	2^{-6}	$2^{140.46}$	2
42	$\mathcal{L}_2$	2^6	$2^{147.46}$	2^{-2}	$2^{140.46}$	2^5
43	$\mathcal{L}$	0	$2^{146.46}$	2^{-6}	$2^{139.46}$	1
44	$\mathcal{L}_1$	2^5	$2^{145.46}$	2^{-3}	$2^{139.46}$	2^2
45	$\mathcal{L}$	0	$2^{142.46}$	2^{-6}	$2^{135.46}$	1
46	$\mathcal{L}_{30}$	2^6	$2^{142.46}$	2^{-4}	$2^{135.46}$	2^2
47	$\mathcal{L}_{31}$	2^6	$2^{142.46}$	2^{-4}	$2^{135.46}$	2^4
48	$\mathcal{L}_0$	2^6	$2^{142.46}$	2^{-4}	$2^{135.46}$	2^6
49	$\mathcal{L}$	0	$2^{142.46}$	2^{-6}	$2^{135.46}$	1
50	$\mathcal{L}_{29}$	2^6	$2^{142.46}$	1	$2^{135.46}$	2^6
51	$\mathcal{L}$	0	$2^{142.46}$	$2^{-5.83}$	$2^{135.46}$	$2^{0.17}$
52	$\mathcal{L}_{25}$	2^6	$2^{142.46}$	1	$2^{135.46}$	$2^{6.17}$
53	$\mathcal{L}_{26}$	2^6	$2^{142.46}$	1	$2^{135.46}$	$2^{12.17}$
54	$\mathcal{L}$	0	$2^{148.63}$	$2^{-5.67}$	$2^{135.46}$	$2^{6.5}$

I Key Recovery Attack Against 7-Round SPEEDY with Linear Cryptanalysis

(See Fig. 6).

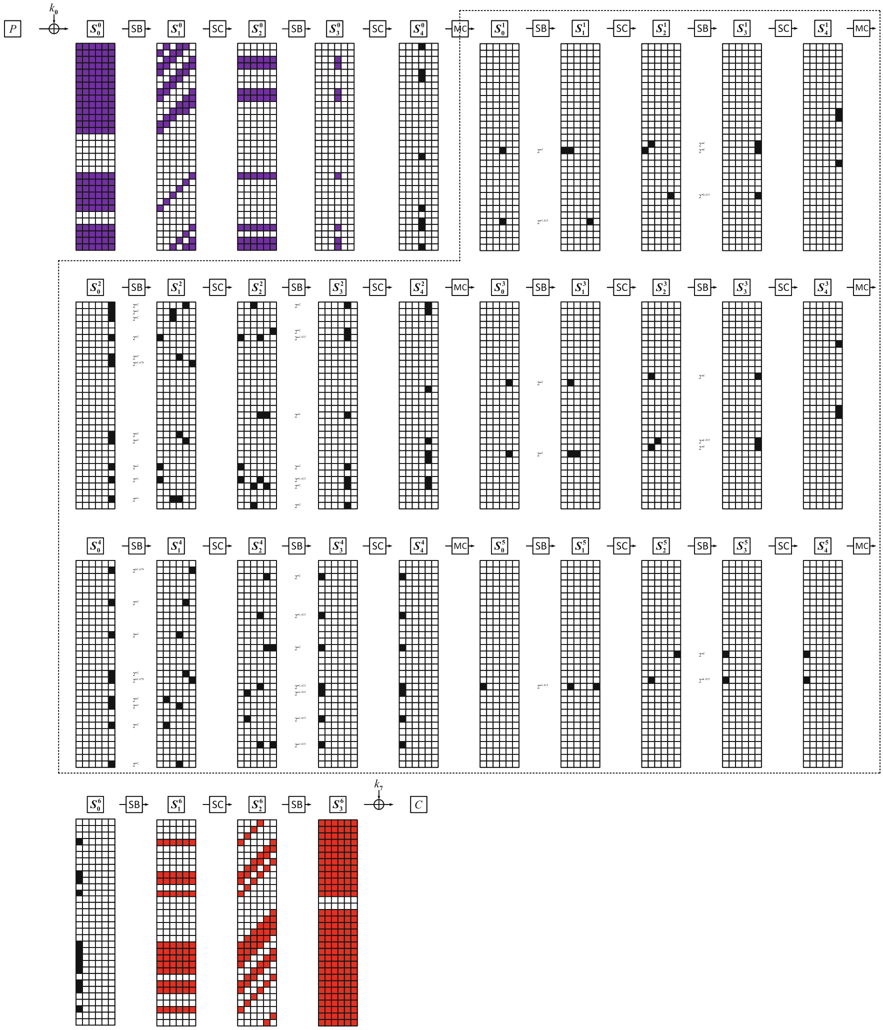


Fig. 6. Key recovery attack on 7-round SPEEDY with linear cryptanalysis.

J Key Recovery Attack against 4-Round SPEEDY with Differential Cryptanalysis

(See Fig. 7).

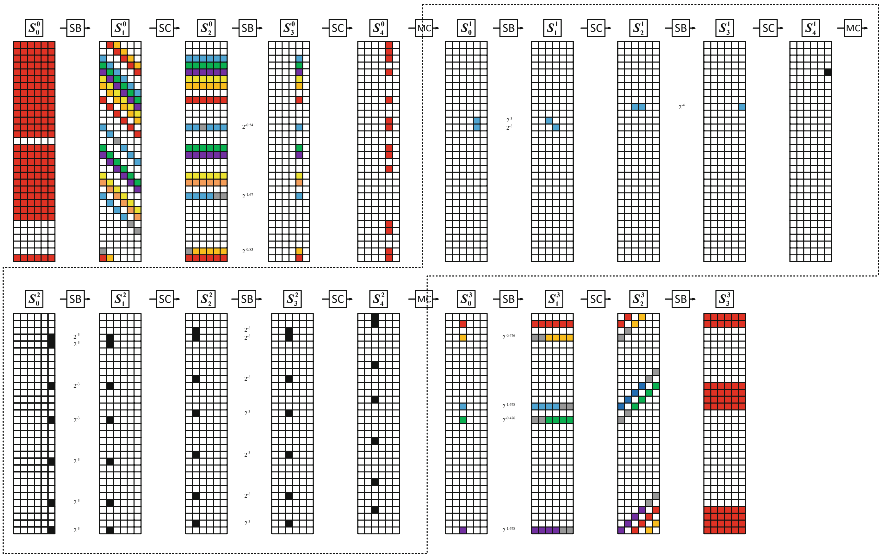


Fig. 7. Key recovery attack on 4-round SPEEDY with differential cryptanalysis.

References

1. Bar-On, A., Dunkelman, O., Keller, N., Weizman, A.: DLCT: a new tool for differential-linear cryptanalysis. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 313–342. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_11
2. Barrett, C., Tinelli, C.: Satisfiability modulo theories. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 305–343. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_11
3. Beierle, C., Leander, G., Todo, Y.: Improved differential-linear attacks with applications to ARX ciphers. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 329–358. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_12
4. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991)
5. Boura, C., David, N., Boissier, R.H., Naya-Plasencia, M.: Better steady than speedy: full break of speedy-7-192. Cryptology ePrint Archive, Paper 2022/1351 (2022)

6. Fan, Y., Li, M., Niu, C., Lu, Z., Wang, M.: Related-tweakey impossible differential attack on reduced-round SKINNY-AEAD M1/M3. In: Galbraith, S.D. (ed.) CT-RSA 2022. LNCS, vol. 13161, pp. 247–271. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-95312-6_11
7. Flórez-Gutiérrez, A., Naya-Plasencia, M.: Improving key-recovery in linear attacks: application to 28-round PRESENT. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 221–249. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_9
8. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 17–25. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_3
9. Leander, G., Moos, T., Moradi, A., Rasoolzadeh, S.: The speedy family of block ciphers - engineering an ultra low-latency cipher from gate level for secure processor architectures. Cryptology ePrint Archive, Paper 2021/960 (2021)
10. Li, M., Hu, K., Wang, M.: Related-tweak statistical saturation cryptanalysis and its application on QARMA. Cryptology ePrint Archive (2019)
11. Liu, Y., Wang, Q., Rijmen, V.: Automatic search of linear trails in ARX with applications to SPECK and chaskey. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 485–499. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_26
12. Longo, G., Zilli, M.V.: Complexity of theorem-proving procedures: some general properties. Revue française d'automatique inf. recherche opé. Inf. théorique **8**(R3), 5–18 (1974)
13. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_33
14. Niu, C., Li, M., Sun, S., Wang, M.: Zero-correlation linear cryptanalysis with equal treatment for plaintexts and tweakeys. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 126–147. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75539-3_6
15. Qin, L., Dong, X., Wang, X., Jia, K., Liu, Y.: Automated search oriented to key recovery on ciphers with linear key schedule: applications to boomerangs in SKINNY and ForkSkinny. IACR Trans. Symmetric Cryptol. **2021**(2), 249–291 (2021)
16. Rohit, R., Sarkar, S.: Cryptanalysis of reduced round speedy. Cryptology ePrint Archive (2022)
17. Selçuk, A.A.: On probability of success in linear and differential cryptanalysis. J. Cryptol. **21**(1), 131–147 (2008)
18. Zhou, C., Zhang, W., Ding, T., Xiang, Z.: Improving the MILP-based security evaluation algorithm against differential/linear cryptanalysis using a divide-and-conquer approach. Cryptology ePrint Archive (2019)

Reconsidering Generic Composition: The Modes A10, A11 and A12 are Insecure

Francesco Berti(✉)

Bar-Ilan University, 529002 Ramat-Gan, Israel
francesco.berti@biu.ac.il

Abstract. Authenticated Encryption (AE) achieves privacy and authenticity with a single scheme. It is possible to obtain an AE scheme gluing together an encryption scheme (privacy secure) and a Message Authentication Code (authenticity secure). This approach is called *generic composition* and its security has been studied by Namprempre et al. [20]. They looked into all the possible gluings of an encryption scheme with a secure MAC to obtain a nonce-based AE-scheme. The encryption scheme is either IV-based (that is, with an additional random input, the initialization vector [IV]) or nonce-based (with an input to be used once, the *nonce*). Nampremepre et al. assessed the security/insecurity of all possible composition combinations except for 4 (N4, A10, A11 and A12). Berti et al. [9] showed that N4 is insecure and that the remaining modes (A10, A11, and A12) are either all secure or insecure.

Here, we prove that these modes are all insecure with a counter-example.

Keywords: AE · generic composition · integrity

1 Introduction

Privacy and authenticity are two of the most important goals of cryptography. Encryption schemes provide privacy, that is, no information about a plaintext (except its length) can be obtained from a ciphertext encrypting it; while Message Authentication Codes (MAC) provide authenticity, that is, it is not possible to send a message impersonating another person. *Authenticated Encryption* (AE) is the cryptographic primitive that provides both. In addition, AE allows the presence of Associated Data (AD), which are data sent in clear but authenticated. This primitive is the object of flourishing research from the seminal papers [3,4,24,26], with many constructions proposed, see for example [11–13,15,23,25] and the CAESAR competition [1,7]. Moreover, there is an ongoing NIST competition for a lightweight AE scheme [21], whose finalists have been announced [22] and whose winner is ASCON [14]. See [16] for a survey of the AE-literature.

F. Berti–Work done when this author was at TU Darmstadt, Germany, CAC - Applied Cryptography.

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 157–176, 2023.
https://doi.org/10.1007/978-3-031-35486-1_8

It is possible to design an AE-scheme from scratch (as the case of OCB [25], for example) or to combine an encryption scheme with a MAC. This second approach is called *generic composition* [3].

About generic composition, the first result is the well-known "Encrypt-then-MAC is secure" [6,18]. Namprempre et al. [20] studied thoroughly the generic composition problem. They realised that while the first result [6,18] assumed that the encryption scheme is probabilistic, the literature moved to IV-based or nonce-based encryption schemes [17,26]. Since probabilistic encryption schemes are hard to design, we usually use a deterministic encryption scheme and provide the random coins needed externally with an initialization vector, the IV [2]. These are the IV-based encryption schemes.

Unfortunately, in practice, the IV is not always sampled as it should, that is, uniformly at random [26]. Thus, we can replace the IV with a *nonce* ("number used once"). Nonce-based encryption schemes are assumed to be secure as long as the nonce is not repeated [26].

Namprempre et al. [20] studied all possible combinations of IV-based and nonce-based encryption schemes with prf-MACs (that is, MACs which provide a pseudo-random tag) to obtain a nonce-based AE scheme. They proved that 164 modes are insecure, 12 secure (9 with IV-based encryption schemes and 3 with nonce-based encryption schemes). Only 4 modes remained elusive: N4 (using a nonce-based encryption scheme) and A10, A11, and A12 (using an IV-based), see Fig. 1. For these modes, the security remained undecided.

Note that all these modes follow the MAC-then-Encrypt paradigm. Moreover, N4, A11, and A12 are among the "most efficient" AE-composition modes, in the sense that they use the nonce, the AD, and the message the least possible number of times (and there is the hope that they are secure).

Finally, their security has been proved using an additional hypothesis: the "Knowledge-of-Tags" (KOT) [20]. However, the problem of knowing if KOT is implied by the privacy requirement of the encryption scheme remains.

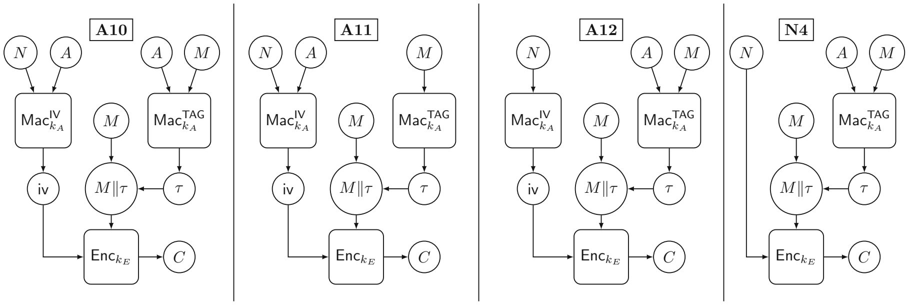

A10
A11
A12
N4
N
A
M
$\mathsf{Mac}^{\mathsf{IV}}_{k_A}$
$\mathsf{Mac}^{\mathsf{TAG}}_{k_A}$
iv
τ
$M\|\tau$
Enc_{k_E}
C

Fig. 1. The modes A10, A11, A12 and N4 [20]. Note that the IV used may be sent in clear along with the ciphertext to speed decryption.

Berti et al. [9] investigated the security of these 4 modes, giving some partial results. First, they proved that N4 is not secure, offering a counterexample with a nonce-based encryption scheme Π with "a kind of Trojan injected". Unfortunately, Π outputs ciphertexts longer than the plaintext. Second, they proved that modes A10, A11, and A12 have the same security: from a counterexample against any of them, we can build counterexamples against the other 2 modes. Third, they proved that the modes A10, A11, and A12 are secure if the secure encryption scheme has any of these two hypotheses: either "misuse-resistance" (that is, using the same IV and different messages, the encryption schemes still outputs pseudorandom ciphertexts) or "message-malleability" (that is, having the encryption of any message with a given IV iv, the adversary can correctly encrypt all other messages with that iv). Since these two hypotheses are, in a certain way, one the opposite of the other, it seems that these modes are secure, but we still do not have the proof.

Our Contribution. Surprisingly, this paper proves that modes A10, A11, and A12 are not secure in general. In particular, they do not provide authenticity. That is, being able to encrypt messages, it is possible to produce a triple (nonce, AD, ciphertext), (N^*, A^*, C^*) which is fresh and valid (that is, $\mathsf{ADec}(N^*, A^*, C^*)$ does not answer "invalid"). We exhibit a counterexample: a secure ivE scheme Π_1, whose composition according to mode A12 with a secure prf-MAC, we can forge. Using [9], we immediately extend this result to modes A10 and A11.

Π_1 uses a tweakable block-cipher[1] (TBC) F. If the message m is s.t. the first two blocks are different, (that is, $m_1 \neq m_2$), substantially Π_1 is a TBC-based version of CTR, that is $c_i = \mathsf{F}^{1,i}(\mathsf{iv}) \oplus m_i$ with the difference that the last block (the one carrying the tag in A12) is encrypted with a slightly different tweak, that is, $c_l = \mathsf{F}^{2,l}(\mathsf{iv}) \oplus m_i$. If the first two message blocks are equal, instead, we modify the encryption of the two last ciphertext blocks: the second-to-last ciphertext block is obtained as $c_{l-1} = \mathsf{F}^{2,m_3}(\mathsf{iv}) \oplus \mathsf{F}^{2,m_3}(m_1) \oplus m_{l-1}$, while the last block is obtained as $c_l = \mathsf{F}^{1,l}(\mathsf{iv}) \oplus m_l$. Further, we assume that Π_1 outputs the IV iv it uses with the ciphertext c. Π_1 is IV-secure, as we prove in Theorem 1.

When mode A12 is implemented with Π_1, a forgery can be created, proceeding as follow: the attacker asks the encryption of a message $M = M_1, ..., M_l$ with nonce N and AD A, obtaining $\mathsf{iv}_1, C_1, ..., C_{l+1}$ (Remind that $C = c = \mathsf{Enc}(\mathsf{iv}, m)$ with $m = M \| \tau$). Then, she asks the encryption C' of N', A', M', where M' is one block longer than M and $M' = \mathsf{iv}, \mathsf{iv}, l+1, ...,$. Our goal is to produce C^* s.t. $\mathsf{ADec}(N', A, C^*) = M$. From C' it is easy to compute the correct $C_1^*, ..., C_l^*$, while to compute C_{l+1}^* (the block encrypting the tag τ), we need both C_{l+1} and C'_{l+1}. The details are in Sect. 4.1.

Since to obtain the correct C_l^*, the adversary needs to know the IV iv used by Π_1 to produce C, a natural solution seems to use the new syntax introduced by Bellare et al. [5]. They assumed that the decryption algorithm needs only to know the ciphertext (and the key) to decrypt correctly (and not the IV, or the

[1] Tweakable block ciphers (TBCs) were introduced by Liskov et al. [19]. They are block-ciphers (BCs) with an additional input, the *tweak*, to add flexibility.

nonce). Unfortunately, this simple solution does not work. In fact, we offer as a counterexample Π_2, a variant of Π_1, where the IV is sent as $C_0 = \mathsf{F}^{0,0}(\mathsf{iv})$.

Third, we show that, to prove that N4 is not secure, we do not need an encryption scheme outputting ciphertexts longer than the plaintexts. We offer two counterexamples: a variant of Π_1 and a variant of the scheme Π presented in [9]. Since TBCs can be build from BCs [19], our construction can be built only from BCs.

This work concludes the classification of all generic composition modes (when the encryption scheme is either nonce-based or IV-based). Moreover, we have proved that IV-security does not imply KOT.

2 Background

Notations. We denote with $\{0,1\}^n$ the set of all n-bit long strings and with $\{0,1\}^*$ the set of all finite strings. We denote the length of the string x with $|x|$. To denote that x is picked uniformly at random from the set $\mathcal{X}$, we use $x \overset{\$}{\leftarrow} \mathcal{X}$.

In our security games, we use *adversaries*, which are probabilistic algorithms. An adversary A who has access to oracles $\mathsf{O}_1, \ldots, \mathsf{O}_T$ is denoted with $\mathsf{A}^{\mathsf{O}_1(\cdot),\ldots,\mathsf{O}_T(\cdot)}$. A $(q_1, \ldots, q_T, t)$*-adversary* A can do at most q_i queries to oracle O_i and runs in time bounded by t. We denote with $\mathsf{A}^{\mathsf{O}_1(\cdot),\ldots,\mathsf{O}_T(\cdot)} \Rightarrow x$ the fact that the adversary A outputs x.

2.1 Tweakable Blockciphers (TBCs)

Encryption schemes and MACs usually use (tweakable)-block ciphers to produce the randomness they need. Formally,

Definition 1. *A* tweakable blockcipher *(*TBC*) is a function* $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ *s.t.* $\forall (k, tw) \in \mathcal{K} \times \mathcal{TW}, \mathsf{F}(k, tw, \cdot) : \{0,1\}^n \rightarrow \{0,1\}^n$ *is a permutation.*

We use often $\mathsf{F}_k^{tw}(x)$ and $\mathsf{F}_k(tw, x)$ to denote $\mathsf{F}(k, tw, x)$. To denote the inverse of $\mathsf{F}_k^{tw}(\cdot)$, we use $\mathsf{F}_k^{-1,tw}(\cdot)$. We call n the *block-length* of F.

We want that a TBC outputs values indistinguishable from random ones. Formally:

Definition 2. *A* TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ *is a* (q, t, ϵ)-tweakable pseudorandom permutation (tprp) *if* $\forall$ (q, t)*-adversary* A*, the following advantage*

$$\left|\Pr[\mathsf{A}^{\mathsf{F}_k(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathsf{A}^{\mathsf{f}(\cdot,\cdot)} \Rightarrow 1]\right| \leq \epsilon$$

where $k \overset{\$}{\leftarrow} \mathcal{K}$ *and* $\mathsf{f} \overset{\$}{\leftarrow} \mathcal{TWP}$*.* $\mathcal{TWP}$ *is the set of all tweakable permutations* f*, that is, the functions* $\mathsf{f} : \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ *s.t.* $\forall tw \in \mathcal{TW}$, $\mathsf{f}(tw, \cdot) : \{0,1\}^n \rightarrow \{0,1\}^n$ *is a permutation.*

When the adversary, even having access also to the inverse of F, cannot distinguish F from f, we say that F is a strong tprp. Formally:

Definition 3. *A* TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ *is a* (q, t, ϵ)-strong tweakable pseudorandom permutation (stprp) *if* $\forall$ (q_1, q_2, t)*-adversary* A*, the following advantage*

$$\left|\Pr[\mathsf{A}^{\mathsf{F}_k(\cdot,\cdot),\mathsf{F}_k^{-1}(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathsf{A}^{\mathsf{f}(\cdot,\cdot),\mathsf{f}^{-1}(\cdot,\cdot)} \Rightarrow 1]\right| \leq \epsilon$$

where $k \xleftarrow{\$} \mathcal{K}$*,* $\mathsf{f} \xleftarrow{\$} \mathcal{TWP}$*,* $\mathsf{f}^{-1}(\cdot,\cdot)$ *is the inverse of* f*, and* $q_1 + q_2 \leq q$*.*

When we do not need that F is a permutation, we use the following security definition

Definition 4. *A* (q, t, ϵ)-pseudorandom function (prf) *is a function* $\mathsf{F} : \mathcal{K} \times \{0,1\}^{n'} \to \{0,1\}^n$ *s.t.* $\forall (q, t)$*-adversary* A*, the following advantage*

$$\left|\Pr[\mathsf{A}^{\mathsf{F}_k(\cdot)} \Rightarrow 1] - \Pr[\mathsf{A}^{\mathsf{f}(\cdot)} \Rightarrow 1]\right| \leq \epsilon$$

where $k \xleftarrow{\$} \mathcal{K}$ *and* $\mathsf{f} \xleftarrow{\$} \mathcal{RF}$ *with* $\mathcal{RF}$ *is the set of all functions* f *which are the functions* $\mathsf{f} : \{0,1\}^{n'} \to \{0,1\}^n$*.*

Note that tprp-secure implies prf-secure [17].

2.2 Encryption Schemes

Encryption schemes are the cryptographic primitive used to provide privacy. To have security, we need that the encryption is probabilistic [17]. Often, to have probabilistic encryption, we use a random input, called the *initialization vector* (IV), or an input used only once, called a *nonce*. Thus, we have IV-based and nonce-based encryption scheme. Formally:

Definition 5. *An* IV-based encryption (ivE) scheme *is a triple* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *where*

- *the* key-generation algorithm Gen *generates a* key k_E *from the keyspace* $\mathcal{K}_E$ *(usually* $k_E \xleftarrow{\$} \mathcal{K}$*);*
- *the* encryption algorithm Enc *takes as input a key* $k_E \in \mathcal{K}_E$*, an initialization vector (*IV*)* iv *in the* IV*-space (*$\mathcal{IV}$*), and a message* m *in the message space* $m \in \mathcal{M}$*, and outputs a string* $c \leftarrow \mathsf{Enc}_{k_E}^{\mathsf{iv}}(m)$ *called* ciphertext*;*
- *the* decryption algorithm Dec *takes as input a key* $k_E \in \mathcal{K}_E$*, an* IV $\mathsf{iv} \in \mathcal{IV}$*, and a ciphertext* $c \in \{0,1\}^*$*, and outputs either a string* $m \in \mathcal{M}$ *or the symbol* $\perp$ *("invalid"); we denote this with* $m/\perp \leftarrow \mathsf{Dec}_{k_E}^{\mathsf{iv}}(c)$*.*

We require that Enc *and* Dec *are the "inverse" of the other. That is,*

- correctness: *if* $\mathsf{Enc}_{k_E}^{\mathsf{iv}}(m) = c$ *(when defined), then,* $\mathsf{Dec}_{k_E}^{\mathsf{iv}}(c) = m$*;*
- tidyness: *if* $\mathsf{Dec}_{k_E}^{\mathsf{iv}}(c) = m \neq \perp$*, then,* $\mathsf{Enc}_{k_E}^{\mathsf{iv}}(m) = c$*.*

We assume that the length of the ciphertexts does not depend on the key and on the IV*, that is,* $\forall m \in \mathcal{M}$ $|\mathsf{Enc}^{\mathsf{iv}}_{k_E}(m)| = |\mathsf{Enc}^{\mathsf{iv}'}_{k'_E}(m)|$ $\forall k_E, k'_E \in \mathcal{K}_E,$ $\mathsf{iv}, \mathsf{iv}' \in \mathcal{IV}$*.*

A nonce-based encryption scheme (nE) *is defined as an* IV*-based encryption scheme where the* IV iv *is replaced with a nonce* n*.*

To distinguish nonce from block-size, we use always capital letters for nonces, e.g. N.

Note that syntactically, ivE and nE schemes are the same. But, their security definitions are different: we want that the ciphertexts are indistinguishable from random when the IVs are randomly picked (for ivE) or used only once (for nE). Formally:

Definition 6. *An* ivE *encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* (q, t, ϵ)*-secure (*ivE*) if* $\forall (q, t)$*-adversary* A*, the following advantage*

$$\left|\Pr[\mathsf{A}^{\mathsf{Enc}^{\$}_{k_E}(\cdot)} \Rightarrow 1] - \Pr[\mathsf{A}^{\$(\cdot)} \Rightarrow 1]\right| \leq \epsilon$$

where $k_E \leftarrow \mathsf{Gen}$*,* $\mathsf{Enc}^{\$}_{k_E}(m)$*, first, randomly picks the* IV*,* $\mathsf{iv} \overset{\$}{\leftarrow} \mathcal{IV}$ *and then outputs* $c \leftarrow \mathsf{Enc}^{\mathsf{iv}}_{k_E}(m)$*, and* $\$$ *picks* $(\mathsf{iv}, c) \overset{\$}{\leftarrow} \mathcal{IV} \times \{0,1\}^{|\mathsf{Enc}^{\$}_{k_E}(m)|}$ *uniformly at random.*

Note that iv is picked in the same way in both cases.

Definition 7. *An* nE *encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* (q, t, ϵ)*-secure (*nE*) if* $\forall (q, t)$*-adversary* A*, the following advantage*

$$\left|\Pr[\mathsf{A}^{\mathsf{Enc}_{k_E}(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathsf{A}^{\$(\cdot,\cdot)} \Rightarrow 1]\right| \leq \epsilon$$

where $k_E \leftarrow \mathsf{Gen}$*, and* $\$$ *picks* $c \overset{\$}{\leftarrow} \{0,1\}^{|\mathsf{Enc}_{k_E}(N,m)|}$ *uniformly at random. The adversary is not allowed to do a query on input* (N, m) *if she has already done a query on input* (N, m') *for* $m \neq m'$*. That is, each nonce* N *is used at most once.*

For both ivE and nE-security, the adversary cannot query the decryption oracle (or an ideal counterpart).

2.3 Message Authentication Codes (MAC)

Message authentication codes (MACs) are the cryptographic primitive used for authenticity.

Definition 8. *A* MAC *is a triple* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *where*

- *the* key-generation algorithm Gen *generates a* key k_A *from the keyspace* $\mathcal{K}_\mathsf{A}$ *(usually* $k_A \overset{\$}{\leftarrow} \mathcal{K}_\mathsf{A}$*);*
- *the* tag-generation algorithm Mac *takes as input a key* $k_A \in \mathcal{K}_\mathsf{A}$*, and a value* x *in the domain space* $x \in \mathcal{X}$*, and outputs a string called* tag $\tau \leftarrow \mathsf{Mac}_{k_A}(x)$*;*

- *the* verification algorithm Vrfy *takes as input a key* $k_A \in \mathcal{K}_{\mathsf{A}}$*, a value* $x \in \mathcal{X}$ *and a tag* τ*, and outputs either a string* $\top$ *("valid") or the symbol* $\bot$ *("invalid") and we denote this with* $\top/\bot \leftarrow \mathsf{Vrfy}_{k_A}(x, \tau)$*.*

We require that Mac *and* Vrfy *are one the "inverse" of the other. That is,*

- correctness: *if* $\mathsf{Mac}_{k_A}(x) = \tau$ *(when defined), then,* $\mathsf{Vrfy}_{k_A}(x, \tau) = \top$*;*
- tidyness: *if* $\mathsf{Vrfy}_{k_A}(x, \tau) = \top$*, then,* $\mathsf{Mac}_{k_A}(x) = \tau$*.*

The tidiness is implied, when the verification algorithm is the most obvious: on input (x, τ), Vrfy_{k_A} computes $\tau' = \mathsf{Mac}_{k_A}(x)$ and checks if $\tau = \tau'$.

The security definition that we use for MAC, as in [20], is not standard: we ask that Mac is a prf. Formally,

Definition 9. *A* MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is* (q, t, ϵ)-prf secure *if* Mac *is a* (q, t, ϵ)-prf *where the key is picked according to* Gen*.*

The standard definition (unforgeability, see [17]) is implied by this definition, but it is not " a suitable starting point when the goal is to create a nAE scheme " [20].

2.4 Authenticated Encryption (AE)

Authenticated Encryption is the cryptographic primitive used to provide both privacy and authenticity. We assume, following [24], that there is a nonce, and there are data to be authenticated but not encrypted. They are called *Associated Data* (AD).

Definition 10. *A* nonce-based authenticated encryption (nAE) *is a triple* $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ *where*

- *the* key-generation algorithm Gen *generates a* key K *from the keyspace* $\mathcal{K}_{\mathsf{AE}}$ *(usually* $K \xleftarrow{\$} \mathcal{K}_{\mathsf{AE}}$*);*
- *the* encryption algorithm AEnc *takes as input a key* $K \in \mathcal{K}_{\mathsf{AE}}$*, a nonce* N *in the nonce-space (*$\mathcal{N}$*), an associated data* A *in the associated data space (*$\mathcal{A}$*), and a message* M *in the message space* $M \in \mathcal{M}_{\mathsf{AE}}$*, and outputs a string* $C \leftarrow \mathsf{AEnc}_K(N, A, M)$ *called* ciphertext*;*
- *the* decryption algorithm ADec *takes as input a key* $K \in \mathcal{K}_{\mathsf{AE}}$*, a nonce* $N \in \mathcal{N}$*, and a ciphertext* $C \in \{0,1\}^*$*, and outputs either a string* $M \in \mathcal{M}_{\mathsf{AE}}$ *or the symbol* $\bot$ *("invalid"); we denote this with* $M/\bot \leftarrow \mathsf{ADec}_K(N, A, C)$*.*

We require that AEnc *and* ADec *are one the "inverse" of the other. That is,*

- correctness: *if* $\mathsf{AEnc}_K(N, A, M) = C$ *(when defined), then,* $\mathsf{ADec}_K(N, A, C) = M$*;*
- tidyness: *if* $\mathsf{ADec}_K(N, A, C) = M \neq \bot$*, then,* $\mathsf{AEnc}_K(N, A, M) = C$*.*

We assume that the length of the ciphertext does not depend on the key K*, that is,* $\forall (N, A, M) \in \mathcal{N} \times \mathcal{A} \times \mathcal{M}_{\mathsf{AE}}$ $|\mathsf{AEnc}_K(N, A, M)| = |\mathsf{AEnc}_{K'}(N, A, M)|$ $\forall K, K' \in \mathcal{K}_{\mathsf{AE}}$*.*

Note that, syntactically, nAE schemes are very similar to nE schemes (Definition 5) with the addition of associated data.

To make the reading clearer, we use capital letters (e.g., M) for the inputs of AEnc and ADec, while small letters (e.g., m) for the inputs of Enc, Dec, Mac, and Vrfy. This will make the next section more accessible.

nAE schemes want to provide privacy and authenticity with the same scheme. The following definition captures this:

Definition 11. *An* nAE *encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{AEnc}, \mathsf{ADec})$ *is* (q_1, q_2, t, ϵ)*-secure (*nAE*) if* $\forall (q_1, q_2, t)$*-adversary* A*, the following advantage*

$$\left| \Pr[\mathsf{A}^{\mathsf{AEnc}_K(\cdot,\cdot,\cdot),\mathsf{ADec}_K(\cdot,\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathsf{A}^{\$(\cdot,\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1] \right| \leq \epsilon$$

where $K \leftarrow \mathsf{Gen}$*,* $\$(N, A, M)$ *outputs a random string with the same length as* $\mathsf{AEnc}_K(N, A, M)$*, and* $\perp(\cdot,\cdot,\cdot)$ *always outputs* $\perp$*. The adversary is not allowed to ask her second oracle on input* (N, A, C) *if she has received* C *as an answer to a query to the first oracle on input* (N, A, M) *for any* $M \in \mathcal{M}_{\mathsf{AE}}$*. Moreover, the adversary cannot query her first oracle on input* (N, A, M) *if she has already queried her first oracle on input* (N, A', M')*. That is, each nonce* N *is used at most once during "encryption" (first oracle) queries.*

This notion implies that the adversary cannot find a *forgery*, that is a triple (N, A, C) which is *fresh* and *valid*, that is, (N, A, C) does not come as answer to a previous query on input (N, A, M) $[C = \mathsf{AEnc}_K(N, A, M)]$ and $\mathsf{ADec}_K(N, A, C) \neq \perp$.

3 Generic Composition and the Elusive Generic Composition Modes

3.1 Generic Composition

A natural way to obtain an AE scheme is to compose an encryption scheme with a MAC [3]. This approach is the so-called *generic composition*. In the original paper considering the security of the generic composition [3], the authors studied the composition of a *probabilistic* encryption schemes[2] with a MAC. There are three possible composition methods: *Encrypt-and-*MAC, MAC*-then-Encrypt*, and *Encrypt-then-*MAC. They proved that Encrypt-then-MAC is always secure.

Namprempre et al. [20] studied the generic composition when the encryption scheme is either ivE or nE-based and the MAC scheme is prf-secure. For ivE-based, the prf-MAC provides both the IV to the ivE scheme and the tag. To prevent trivial attacks, there is the domain separation between these two calls, that is, the IV iv is obtained from $\mathsf{Mac}^{\mathsf{IV}}_{k_A}$, while the tag τ from $\mathsf{Mac}^{\mathsf{TAG}}_{k_A}$. There are three possible type compositions modes, with $C = \mathsf{AEnc}_K(N, A, M)$ with $K = (k_E, k_A)$:

[2] A probabilistic encryption scheme is a triple $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ s.t. the output of Enc is probabilistic. For all its other requirements, see [17].

E&M Encrypt-&-MAC where $C = (c\|\tau)$, $c = \mathsf{Enc}^{\mathsf{iv}}_{k_E}(M)$, $\mathsf{iv} = \mathsf{Mac}^{\mathsf{IV}}_{k_A}(N|U, A|U, M|U)$ and $\tau = \mathsf{Mac}^{\mathsf{TAG}}_{k_A}(N|U, A|U, M|U)$. (With $X|U$, we denote that the input either contains the string X or is absent).

EtM Encrypt-then-MAC, where $C = (c\|\tau)$, $c = \mathsf{Enc}^{\mathsf{iv}}_{k_E}(M)$, $\mathsf{iv} = \mathsf{Mac}^{\mathsf{IV}}_{k_A}(N|U, A|U, M|U)$ and $\tau = \mathsf{Mac}^{\mathsf{TAG}}_{k_A}(N|U, A|U, C)$.

MtE MAC-then-Encrypt , where $C = c$, $c = \mathsf{Enc}^{\mathsf{iv}}_{k_E}(m)$, with $m = M\|\tau$, $\mathsf{iv} = \mathsf{Mac}^{\mathsf{IV}}_{k_A}(N|U, A|U, M|U)$ and $\tau = \mathsf{Mac}^{\mathsf{TAG}}_{k_A}(N|U, A|U, C)$.

These are the so called A-modes.

In general, we can suppose that the IV is public and it is sent with C. This can speed decryption (anyway, we can check if the IV is correct). The fact that the IV is public follows from [20]'s description.

When we compose a MAC with an nE scheme, then, we have the following types of composition modes, $C = \mathsf{AEnc}_K(N, A, M)$ with $K = (k_E, k_A)$:

E&M Encrypt-&-MAC where $C = (c\|\tau)$, $c = \mathsf{Enc}^{N}_{k_E}(M)$, $\tau = \mathsf{Mac}^{\mathsf{TAG}}_{k_A}(N|U, A|U, M|U)$.

EtM Encrypt-then-MAC, where $C = (c\|\tau)$, $c = \mathsf{Enc}^{N}_{k_E}(M)$, and $\tau = \mathsf{Mac}^{\mathsf{TAG}}_{k_A}(N|U, A|U, C)$.

MtE MAC-then-Encrypt , where $C = c$, $c = \mathsf{Enc}^{N}_{k_E}(m)$, with $m = M\|\tau$, and $\tau = \mathsf{Mac}^{\mathsf{TAG}}_{k_A}(N|U, A|U, C)$.

These are the so-called N-modes.

Note that both AEnc and Enc use the same nonce.

Thus, there are 160 possible modes when we use an ivE scheme and 20 possible modes when we use a nE scheme.

3.2 The Four Elusive Modes: A10, A11, A12, N4

Namprempre et al. [20] were able to prove the security of 9 modes for ivE-composition and 3 for nE-composition, and the insecurity of all others except for 4 modes, all MAC-then-Encrypt type:

A10 MtE with $\mathsf{MAC}^{\mathsf{IV}}(N, A, U)$ and $\mathsf{MAC}^{\mathsf{TAG}}(U, A, M)$.
A11 MtE with $\mathsf{MAC}^{\mathsf{IV}}(N, A, U)$ and $\mathsf{MAC}^{\mathsf{TAG}}(U, U, M)$.
A12 MtE with $\mathsf{MAC}^{\mathsf{IV}}(N, U, U)$ and $\mathsf{MAC}^{\mathsf{TAG}}(U, A, M)$.
N4 MtE with $\mathsf{MAC}^{\mathsf{TAG}}(U, A, M)$.

We have depicted them in Fig. 1.

For decryption either the IV is sent in clear and it is checked and used for decryption, or it is recomputed from $(N, A|U)$.

Knowledge-of-Tag Based Security. Namprempre et al. [20] proved that modes A10, A11, and A12 are secure if the ivE-scheme is Knowledge-of-Tag-secure (KOT). In the KOT-experiment, "*knowing* a tag is captured by introducing a plaintext extractor Ext, a deterministic algorithm that takes as input all the inputs explicitly available to the forging adversary and outputs a string x or $\perp$" [20]. Roughly speaking, a scheme is KOT-secure, if the adversary cannot " produce a forgery that uses an old $\mathsf{iv}^* = \mathsf{iv}_j$ and an old $m^* \| \tau^* = m_i \| \tau_i$, for which it [the adversary] does not (explicitly) know τ_i, and yet the extractor fails to determine this $m_i \| \tau_i$. Loosely speaking if the forger wins the KOT game, it has done so without (extractable) knowledge of the tag τ_i" [20]. We depict the experiment in the extended version [8] of the paper.

It was left open the problem of whether ivE-security implies KOT.

Partial Results on These Modes [9]**.** At Indocrypt18, Berti et al. [9] proved some results about these modes: 1) mode N4 is insecure (using an nE-scheme which expands the ciphertext), 2) modes A10, A11, and A12 are either all secure or insecure, 3) modes A10, A11, A12 are secure if the IV scheme used is either misuse resistant or "message-malleable". On the other hand, if the ivE scheme used is either stateful or untidy, the modes are not secure. Here, we give some insights into these results.

Mode N4 is Insecure. Berti et al. [9] provides a counterexample using the nE scheme Π (detailed also in the extended version). Π has a key composed of two components $k_E = (k, v^*)$ where k is a key for a TBC with n-bit block, and v^* is a n-bit random string.

For the encryption Π proceeds as follow: the first ciphertext block c_0 is a pseudorandom value, except if the nonce is 1. In this case $c_0 = v^*$, where v^* is a secret random value; all others ciphertext block (except the last) are computed as $c_i = \mathsf{F}_k^{i,0}(N) \oplus m_i$, the last ciphertext block is computed as $c_l = \mathsf{F}_k^{l,0}(N) \oplus m_l$, except if the nonce is either 1 or 2 and the second to last message block m_{l-1}, is v^*: in this case, $c_l = \mathsf{F}_k^{l,1}(0) \oplus m_l$. That is, m_l is encrypted in the same way with both $N = 1$ and $N = 2$ in the case $m_{l-1} = v^*$.

We leave the proof that this scheme is nE-secure to the original paper [9],[3] as well with the description when the length of the message is not a multiple of n.

Observe that the ciphertext is n-bit longer than the message since there is the block c_0. We can see v^* as the trigger of a trojan which forces the same block to be encrypted in the same way in two different encryption queries.

The forgery against N4, when Enc is implemented with Π is straightforward:

- Authenticated encrypt $(1, A, M)$ with $M = M_1, ..., M_l$, obtaining C. Note that $C_0 = v^*$.
- Authenticated encrypt $(2, A, M^1)$ with $M^1_{l-1} = v^*$ and $|M| = |M^1|$, obtaining C^1.
- The forgery is $(1, A, C^*)$ with $C^*_0 = v^*$, $C^*_i = C_i \oplus M_i \oplus M^1_i$ for $i = 1, ..., l$, and $C^*_{l+1} = C^1_{l+1}$ [we remind that C^*_{l+1} encrypts the tag in N4 when Enc is Π]. Note that $\mathsf{ADec}(1, A, C^*) = M^1$.

The forgery is correct (we leave the easy proof to the original paper).

[3] The only problem is if the adversary can do an encryption query (N, m) with $N = 1$ and $m_{l-1} = v^*$, but this cannot happen since v^* is random and leaked only during a query with $N = 1$.

Equivalent Security Among Modes A10, A11 and A12. In the same paper, Berti et al. [9] proved that modes A10, A11, and A12 are either all secure or all insecure. First, they proved that all forgeries (except with negligible probability) must use an IV iv and a tag τ already computed. Then, they prove that in this case (reusing an iv and a τ) if an adversary can create a forgery against one of these modes, she can easily create a forgery against the other two modes. The main ingredients of this last step are these:

- *A12 secure ⇒ A10 secure:* Since the nonce N cannot be repeated during encryption queries, the adversary cannot distinguish if $\mathsf{iv} = \mathsf{MAC}^{\mathsf{IV}}_{k_A}(N)$ or $\mathsf{iv} = \mathsf{MAC}^{\mathsf{IV}}_{k_A}(N, A)$.
- *A11 secure ⇒ A10 secure:* Encrypt with A11 $M' = \mathsf{H}(A)\|M$, with H a hash function. Use an ivE scheme for A11 s.t. the encryption of $\mathsf{H}(A)$ is independent from the one of M, e.g., $\mathsf{Enc}'_{k_E}(\mathsf{iv}, M') = \mathsf{f}(k_E, \mathsf{iv}) \oplus \mathsf{H}(A)\|\mathsf{Enc}_{k_E}(\mathsf{iv}, M)$, where f is a random function $\mathsf{f} : \{0,1\}^{2n} \to \{0,1\}^n$.
- *A10 secure ⇒ A12 secure:* We use the same idea as before, encrypting with A12 $M' = \mathsf{H}(A)\|M$.
- *A10 secure ⇒ A11 secure:* We use a similar idea, but here we modify the nonce. The nonce used for A10 is N, while for A11 is $N' = N\|\mathsf{H}(A)$.

We leave the full details to the original paper [9] and its extended version [10].

Partial Security/Unsecurity Results. Finally, in the same paper [9], the authors proved that modes A10, A11 and A12 are secure if the ivE scheme is either "*misuse resistant*" (that is, an adversary has no advantage if she can reuse the same IV during encryption queries[4]) or *message-malleable* (that is, if an adversary receives the decryption, different from $\perp$, of (iv, c), she can correctly decrypt (iv, c') $\forall c'$, as for example CTR, Counter mode [17]).

On the other hand, if the ivE scheme is not tidy or stateful, then the adversary can create a forgery against modes A10, A11, and A12 when implemented with certain ivE schemes (for the stateful case, we can use a variant of the scheme used against N4). We leave the details to the original paper [9] and its extended version [10].

4 The Modes A10, A11, A12 are Insecure

Now, we show that mode A12 is insecure, giving a counterexample. Thanks to [9], this means that also modes A11 and A10 are not secure.

The first natural idea is to start from the counterexample against N4 and try to adapt it to the A12 case. But this is impossible because the iv is random, and the adversary does not choose it. Thus, if too many IVs reveal v^* or for which the last block is encrypted differently, the scheme is no more ivE-secure. On the other hand, with too few such IVs, the forgery may be done only with negligible probability.

Thus, we need a different idea.

[4] Note that this misuse-resistant definition is weaker then the standard one (see [26] for the original definition), where the adversary can do also decryption queries.

4.1 Warming up - Suppose that ivE Outputs the IV

We start considering the case when the ivE scheme reveals the IV it used during the encryption queries. Note that in mode A12, the AE scheme does not need to reveal the IV since it can be correctly computed even by the decryption oracle ($\mathsf{iv} = \mathsf{MAC}^{\mathsf{IV}}_{k_A}(N)$). But, following the original paper, we assume that the IV is revealed. This follows also from the KOT definition [20].

Construction. We propose an ivE-scheme Π_1 which is based on a TBC F and whose key k_E is the key k of the TBC.

If the message is s.t. the first two blocks are different, (that is, $m_1 \neq m_2$), substantially it is a TBC-based version of CTR, that is $c_i = \mathsf{F}^{1,i}_k(\mathsf{iv}) \oplus m_i$ with the difference that the last block (the one carrying the tag in A12) is encrypted with a slightly different tweak, that is, $c_l = \mathsf{F}^{2,l}_k(\mathsf{iv}) \oplus m_l$. Instead, if the first two message blocks are equal, the encryption is the same except for the two last ciphertext blocks: the second-to-last ciphertext block is obtained as $c_{l-1} = \mathsf{F}^{2,m_3}_k(\mathsf{iv}) \oplus \mathsf{F}^{2,m_3}_k(m_1) \oplus m_{l-1}$, while the last block is obtained as $c_l = \mathsf{F}^{1,l}_k(\mathsf{iv}) \oplus m_l$. The details are in Algorithm 1.

The idea is that if $m_1 = m_2$, we are encrypting the second to last block (not the last block because it carries the tag that it is not known by an adversary, differently from the message that she has chosen to encrypt) in a secure way. Still, it reveals the information necessary to forge using previous encryptions. Note that if the adversary asks for an encryption of a message M with block-length $l-1$, she receives the iv used to encrypt and a ciphertext C. Now, if she asks to encrypt a second message M' s.t. it has block-length $l' = l+1$, $M_1 = M_2 = \mathsf{iv}$, and $M_3 = l+1$, she receives C', where a random iv' is used. C_{l+1} and C'_{l+1} reveal the crucial information for the forgery:

$$C_{l+1} \oplus C'_{l+1} \oplus M'_{l+1} = \mathsf{F}^{2,l+1}_k(\mathsf{iv}) \oplus m_{l+1} \oplus \mathsf{F}^{2,l+1}_k(\mathsf{iv}') \oplus \mathsf{F}^{2,l+1}_k(\mathsf{iv}) \oplus M'_{l+1} \oplus M'_{l+1} = m_{l+1} \oplus \mathsf{F}^{2,l+1}_k(\mathsf{iv}')$$

where $m = M \| \tau$, thus $m_i = M_i$ for $i = 1, ..., l$ and m_{l+1} is the tag τ of A12.

For simplicity, we have considered the case where all message has a length of a multiple of n with a minimum of $3n$. We can easily extend Π_1 to overcome these limitations.

ivE-*security of* Π_1. The ivE-security of Π_1 is straightforward. It is easy to see that each ciphertext block is obtained XORing at least a call to F that has never been asked before, with the following exceptions:

- if two IVs are repeated, that is $\mathsf{iv}^i = \mathsf{iv}^j$;
- if iv^j is equal to m^i_1 with $i \leq j$;

Algorithm 1. The ivE encryption algorithm Π_1.

It uses a TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ with $\mathcal{TW} = \{1,2\} \times \{0,1\}^n$

```
Gen:
  – Return $k \xleftarrow{\$} \mathcal{K}$

Enc_k(iv, m):
  – Parse $m = m_1, ..., m_l$ with $|m_i| = n$
  – For $i = 1, ..., l-2$
      • $c_i = \mathsf{F}_k^{1,i}(\mathsf{iv}) \oplus m_i$
  – If $m_1 \neq m_2$
      • $c_{l-1} = \mathsf{F}_k^{1,l-1}(\mathsf{iv}) \oplus m_{l-1}$
      • $c_l = \mathsf{F}_k^{2,l}(\mathsf{iv}) \oplus m_l$
  – Else
      • $c_{l-1} = \mathsf{F}_k^{2,m_3}(\mathsf{iv}) \oplus \mathsf{F}_k^{2,m_3}(m_1) \oplus m_{l-1}$
      • $c_l = \mathsf{F}_k^{1,l}(\mathsf{iv}) \oplus m_l$
  – Return $(\mathsf{iv}, c)$ with $c = (c_1, ..., c_l)$

Dec_k(iv, c):
  – Parse $c = c_1, ..., c_l$ with $|c_i| = n$
  – For $i = 1, ..., l-2$
      • $m_i = \mathsf{F}_k^{1,i}(\mathsf{iv}) \oplus c_i$
  – If $m_1 \neq m_2$
      • $m_{l-1} = \mathsf{F}_k^{1,l-1}(\mathsf{iv}) \oplus c_{l-1}$
      • $m_l = \mathsf{F}_k^{2,l}(\mathsf{iv}) \oplus c_l$
  – Else
      • $m_{l-1} = \mathsf{F}_k^{2,m_3}(\mathsf{iv}) \oplus \mathsf{F}_k^{2,m_3}(m_1) \oplus c_{l-1}$
      • $m_l = \mathsf{F}_k^{1,l}(\mathsf{iv}) \oplus c_l$
  – Return $m = (m_1, ..., m_l)$
```

But both conditions happen with negligible probability since the IVs are randomly picked. Note that this the reason why there is a first component of the tweak that it is different for c_l (when $m_1 \neq m_2$), and c_{l-1} (when $m_1 = m_2$). Formally,

Theorem 1. *Let* F *be a* $(q_1, t, \epsilon_{\mathsf{tprp}})$-$\mathsf{tprp}$, *where the block-length is* n *bits, then* Π_1 *is* (q, t, ϵ)-ivE-*secure with*

$$\epsilon \leq \epsilon_{\mathsf{tprp}} + \frac{(\tilde{L}+2)(q+1)^2}{2^{n+1}},$$

where $q_1 = L + q$, *with* L *the total number of message blocks to be encrypted, and* $\tilde{L}$ *the maximal number of blocks in any message query.*

We leave the easy proof to the extended version [8].

Forgery for A12 When the ivE-*Scheme is* Π_1. The idea of the forgery is to ask the encryption of a message M s.t. $M_1 \neq M_2$ and then ask the encryption of a message M' s.t. $M'_1 = M'_2 = \mathsf{iv}$[1] which is at least a block longer than M. For the forgery, we proceed as follow:

- Ask the encryption of (N, A, M) with the message M s.t. $M_1 \neq M_2$ and it has l blocks. Obtain the ciphertext $C = (\mathsf{iv}, C_1, ..., C_l, C_{l+1})$. Π_1 encrypts $m = M \| \tau$ with $\tau = \mathsf{Mac}_{k_A}^{\mathsf{TAG}}(A, M)$ using as IV $\mathsf{iv} = \mathsf{Mac}_{k_A}^{\mathsf{IV}}(N)$.
- Ask the encryption of (N', A', M') with the message M' s.t. $M'_1 = M'_2 = \mathsf{iv}$, $M'_3 = l + 1$ and it has $l + 1$ blocks, and $N \neq N'$. Obtain the ciphertext $C' = (\mathsf{iv}', C'_1, ..., C'_l, C'_{l+1}, C'_{l+2})$.
- The forgery is (N^*, A^*, C^*) with $N^* = N'$, $A^* = A$ and C^* defined as follow:
  - $\mathsf{iv}^* = \mathsf{iv}'$;

- $C_i^* = C_i' \oplus M_i' \oplus M_i$ for $i = 1, ..., l$;
- $C_{l+1}^* = C_{l+1} \oplus C_{l+1}' \oplus M_{l+1}'$.

This is a valid forgery (encrypting M), as we formally prove in the next proposition:

Proposition 1. *Let Π_1 be the* ivE *scheme defined in Algorithm 1. Let* MAC *be a* prf*-secure* MAC *with n-bit long output. Then the A12 composition is not* nAE*-secure.*

Proof. Observe that to break the nAE security (Definition 11) is enough to provide a valid forgery because, in the left world $(\mathsf{AEnc}, \mathsf{ADec})$, the answer will be different from the right world $(\$, \perp)$ which is always invalid.

Now, we have to prove that the forgery just described is *fresh* and *valid*.

We use the same notation as in the previous paragraph.

The fact that (N^*, A^*, C^*) is fresh is trivial since with nonce N^*, we have obtained only a ciphertext C', which is one block longer.

For validity, we start observing that we have never repeated a nonce. Now, we want to prove that $\mathsf{ADec}(N^*, A^*, C^*) = M$. To do this we compute $\tilde{C} = \mathsf{AEnc}(N', A, M)$:

- $\tilde{\mathsf{iv}} := \mathsf{MAC}^{\mathsf{IV}}(N')$. Thus, $\tilde{\mathsf{iv}} = \mathsf{iv}' = \mathsf{iv}^*$;
- For $i = 1, ..., l-2$, $\tilde{C}_i = \mathsf{F}_k^{1,i}(\mathsf{iv}^*) \oplus M_i = \mathsf{F}_k^{1,i}(\mathsf{iv}') \oplus M_i' \oplus M_i' \oplus M_i = C_i' \oplus M_i' \oplus M_i$ (and both M_i and M_i' are known by the adversary since she has chosen them).
- Since $M_1 \neq M_2$, then $\tilde{C}_l = \mathsf{F}_k^{1,l}(\tilde{\mathsf{iv}}) \oplus M_l = \mathsf{F}_k^{1,l}(\mathsf{iv}') \oplus M_l' \oplus M_l' \oplus M_l = C_l' \oplus M_l' \oplus M_l$ (and both M_i and M_i' are known by the adversary since she has chosen them). Note that C_l' is the third to last ciphertext block of C'. In fact, during the second encryption query the message encrypted by Π_1 is $m' = M' \| \tau' = M_1' \| ... \| M_l' \| M_{l+1}' \| \tau'$.
- $\tilde{\tau} = \mathsf{MAC}_{k_A}^{\mathsf{TAG}}(A, M) = \tau$.
- $\tilde{C}_{l+1} = \mathsf{F}_k^{2,l+1}(\tilde{\mathsf{iv}}) \oplus \tilde{\tau} = \mathsf{F}_k^{2,l+1}(\tilde{\mathsf{iv}}) \oplus \mathsf{F}_k^{2,l+1}(\mathsf{iv}) \oplus M_{l+1}' \oplus \mathsf{F}_k^{2,l+1}(\mathsf{iv}) \oplus \tilde{\tau} \oplus M_{l+1}' = \mathsf{F}_k^{2,l+1}(\mathsf{iv}') \oplus \mathsf{F}_k^{2,M_3'}(M_1') \oplus M_{l+1}' \oplus \mathsf{F}_k^{2,l+1}(\mathsf{iv}) \oplus \tau \oplus M_{l+1}' = C_{l+1}' \oplus C_{l+1} \oplus M_{l+1}'$, since $\tilde{\mathsf{iv}} = \mathsf{iv}'$, $M_3' = l+1$, $M_1' = \mathsf{iv}$ and M_{l+1}' is known by the adversary (since chosen).

Thus, $\tilde{C} = C^*$ consequently $\mathsf{ADec}(N^*, A^*, C^*) = \mathsf{ADec}(N^*, A^*, \tilde{C}) = M$.

This and [9] proves that modes A10, A11 and A12 are not nAE-secure. Formally,

Theorem 2. *Let* MAC *be a* prf*-secure* MAC*. Then, there exist 3* ivE*-secure* ivE*-encryption schemes Π_{10}, Π_{11}, Π_{12} outputting the* IV *s.t. modes A10, A11 and A12 are not* nAE*-secure when implemented with* MAC *and the corresponding Π.*

Proof. For mode A12, the proof follows easily from the previous proposition, setting $\Pi_{12} := \Pi_1$ where the TBC has a block-length equal to the size of the MAC output. The proof that Π_{12} is ivE-secure is in Proposition 1.

For the other two cases, A10 and A11, in [9] it has been proved that a forgery against a mode A12 composition can be extended to a forgery to a mode A10 or A11 composition(see Sect. 3.2). This proves our statement.

As a side remark, it is easy to see that if in our forgery attack we had set $A' = A$, Π_1 is a good candidate as Π_{10} and Π_{11}. The details are provided in the extended version [8].

This result also proves a domain separation between ivE and KOT. Formally,

Theorem 3. ivE-*secure* $\not\Rightarrow$ KOT-*secure.*

Proof. Π_1 is ivE-secure and not KOT-secure. The previous attack breaks the KOT-definition.

4.2 Broadcasting the IV in the Ciphertext - Attack When the IV is Hidden

In an interesting paper, Bellare et al. [5] realized that sending the nonce along with the ciphertext can create security problems. Thus, they proposed a new syntax (NBE2) for AE scheme where the decryption oracle needs only as input the ciphertext and the header (and the key) to decrypt correctly.

Note that nonce-based encryption scheme and IV-based encryption scheme are syntactically equivalent (see Sect. 2.2), thus we can use their syntax also for ivE-scheme.

Since in the forgery attack we have presented in the previous section, we need that the adversary knows the IV used by Π_1 during the first authenticated encryption query, it is natural to wonder if it is enough to hide the IV used to prevent the previous attack and prove that modes A10, A11 and A12 are secure. Moreover, the IV is not needed to decrypt since it can be recomputed from N.

Unfortunately, this is not the case, as we prove in this section by providing a variant of Π_1, called Π_2 which can be forged even if the adversary has no clue about the IV used.

The Construction Π_2. We add a block before all the ciphertext, called c_0. This block contains an encryption of the iv used ($c_0 = \mathsf{F}_k^{0,0}(\mathsf{iv})$). Now, even if the adversary cannot recover the iv from c_0, this pseudo-random block can be used in the forgery. Then, Π_2 is equal to Π_1 with the exception of c_{l-1} when $m_1 = m_2$. Instead of computing $c_{l-1} = \mathsf{F}_k^{2,m_3}(\mathsf{iv}) \oplus \mathsf{F}_k^{2,m_3}(m_1) \oplus m_{l-1}$, we compute $c_{l-1} = \mathsf{F}_k^{2,m_3}(\mathsf{iv}) \oplus \mathsf{F}_k^{2,m_3}(w) \oplus m_{l-1}$, with $w = \mathsf{F}_k^{-1,(0,0)}(m_1)$. Thus, with m_1, we can tell the encryption algorithm for which iv, that we do not know, we want some information.

Note that we can create a variant for the decryption that does not need IV as an input: Dec'. Dec' simply computes the iv as $\mathsf{iv} = \mathsf{F}_k^{-1,(0,0)}(c_0)$ and then proceeds as for Dec.

ivE-*Security of* Π_2. We have only to show that the modification that we have done does not affect security. In particular, c_0 is a pseudo-random block, and we need to use a stprp-secure F because we use F^{-1} during encryption, and we have a problem if w is equal to a previous iv.

Thus, we have that

Algorithm 2. The ivE encryption algorithm Π_2.

It uses a TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ with $\mathcal{TW} = \{0,1,2\} \times \{0,1\}^n$

```
Gen:
  – Return $k \xleftarrow{\$} \mathcal{K}$

Enc_k(iv, m):
  – Parse $m = m_1, ..., m_l$ with $|m_i| = n$
  – $c_0 = \mathsf{F}_k^{0,0}(\mathsf{iv})$
  – For $i = 1, ..., l-2$
      • $c_i = \mathsf{F}_k^{1,i}(\mathsf{iv}) \oplus m_i$
  – If $m_1 \neq m_2$
      • $c_{l-1} = \mathsf{F}_k^{1,l-1}(\mathsf{iv}) \oplus m_{l-1}$
      • $c_l = \mathsf{F}_k^{2,l}(\mathsf{iv}) \oplus m_l$
  – Else
      • $w = \mathsf{F}_k^{-1,(0,0)}(m_1)$
      • $c_{l-1} = \mathsf{F}_k^{2,m_3}(\mathsf{iv}) \oplus \mathsf{F}_k^{2,m_3}(w) \oplus m_{l-1}$
      • $c_l = \mathsf{F}_k^{1,l}(\mathsf{iv}) \oplus m_l$
  – Return $c = (c_0, c_1, ..., c_l)$

Dec_k(iv, c):
  – Parse $c = c_0, c_1, ..., c_l$ with $|c_i| = n$
  – If $c_0 \neq \mathsf{F}_k^{0,0}(\mathsf{iv})$
      • Return $\perp$
  – For $i = 1, ..., l-2$
      • $m_i = \mathsf{F}_k^{1,i}(\mathsf{iv}) \oplus c_i$
  – If $m_1 \neq m_2$
      • $m_{l-1} = \mathsf{F}_k^{1,l-1}(\mathsf{iv}) \oplus c_{l-1}$
      • $m_l = \mathsf{F}_k^{2,l}(\mathsf{iv}) \oplus c_l$
  – Else
      • $w = \mathsf{F}_k^{-1,(0,0)}(m_1)$
      • $m_{l-1} = \mathsf{F}_k^{2,m_3}(\mathsf{iv}) \oplus \mathsf{F}_k^{2,m_3}(w) \oplus c_{l-1}$
      • $m_l = \mathsf{F}_k^{1,l}(\mathsf{iv}) \oplus c_l$
  – Return $m = (m_1, ..., m_l)$
```

Theorem 4. *Let* F *be a* $(q_1, t, \epsilon_{\mathsf{stprp}})$-stprp, *where the block-length is* n *bits, then* Π_2 *is* (q, t, ϵ)-ivE-*secure with*

$$\epsilon \leq \epsilon_{\mathsf{stprp}} + \frac{(\tilde{L}+4)(q+1)^2}{2^{n+1}},$$

where $q_1 = L + 3q$, *with* L *the total number of message blocks to be encrypted, and* $\tilde{L}$ *the maximal number of blocks in any message query.*

We leave the easy proof to the extended version [8].

Forgery for A12 When the ivE *Scheme is* Π_2. It is easy to extend to forgery for mode A12 when implemented with Π_1 to mode A12 implemented with Π_2 as follow:

- Ask the encryption of (N, A, M) with the message M s.t. $M_1 \neq M_2$ and it has l blocks. Obtain the ciphertext $C = (C_0, C_1, ..., C_l, C_{l+1})$.
- Ask the encryption of (N', A', M') with the message M' s.t. $M'_1 = M'_2 = C_0$, $M'_3 = l + 1$ and it has $l + 1$ blocks, and $N \neq N'$. Obtain the ciphertext $C' = (C'_0, C'_1, ..., C'_l, C'_{l+1}, C'_{l+2})$.
- The forgery is (N^*, A^*, C^*) with $N^* = N'$, $A^* = A$ and C^* defined as follow:
    - $C^*_0 = C'_0$;

- $C^*_i = C'_i \oplus M'_i \oplus M_i$ for $i = 1, ..., l$;
- $C^*_{l+1} = C_{l+1} \oplus C'_{l+1} \oplus M'_{l+1}$.

This is a valid forgery (encrypting M). Formally,

Proposition 2. *Let Π_2 be the* ivE *scheme defined in Algorithm 1. Let* MAC *be a* prf*-secure* MAC *with n-bit long output. Then the A12 composition is not* nAE*-secure.*

The proof is the same as for Proposition 1 with the difference that we have to replace in the computation of $\tilde{C}_{l+1}$, $\mathsf{F}_k^{2,M'_3}(M'_1)$ with $\mathsf{F}_k^{2,M'_3}(w')$ where

$$w' = \mathsf{F}_k^{-1,(0,0)}(C^*_0) = \mathsf{F}_k^{-1,(0,0)}(M'_0) = \mathsf{F}_k^{-1,(0,0)}\left(\mathsf{F}_k^{(0,0)}(\mathsf{iv})\right) = \mathsf{iv}.$$

This and [9] proves that modes A10, A11 and A12 are not nAE-secure even if the IV is not broadcast. Formally,

Theorem 5. *Let* MAC *be a* prf*-secure* MAC*. Then, there exist 3* ivE*-secure* ivE*-encryption schemes Π_{10}, Π_{11}, Π_{12} s.t 1) they do not output the* IV*, 2) the composition of Π_i with a* prf*-secure* MAC *according to mode A i is not* nAE*-secure for $i = 10,\ 11,\ 12$.*

The proof is the same as for Theorem 2.

Note that this attack proves that ivE-security does not imply Knowledge-of-Tag secure.

4.3 Fixed Length nE Scheme for N4

Finally, we prove that it is unnecessary to use an nE encryption scheme whose ciphertext is longer than plaintext to prove that N4 is not secure. We propose two constructions: one which is a modified version of Π_1 (Algorithm 1) and another is a version of the scheme of [9].

Π_3, a Variant of Π_1. The first idea is to use Π_1 directly since ivE-schemes and nE-schemes are syntactically equivalent.

Unfortunately, Π_1 is not nE-secure. It is trivial to see that the condition iv^i equal to m^j for $j \leq i$ does not happen with negligible probability since the IV is replaced with a nonce which the adversary chooses.

Thus, we modify Π_1, obtaining Π_3 as follows:

- the condition if $m_1 \neq m_2$ becomes $m_1 \neq m_2 \wedge N \neq 2$
- in the else we replace $c_{l-1} = \mathsf{F}_k^{2,m_3}(\mathsf{iv}) \oplus \mathsf{F}_k^{2,m_3}(m_1) \oplus m_{l-1}$ with $c_{l-1} = \mathsf{F}_k^{2,m_3}(N) \oplus \mathsf{F}_k^{2,m_3}(1) \oplus m_{l-1}$

The idea is that we always enter in the if except when the nonce $N = 2$. When we do not enter in the if, we obtain information to obtain a forgery combined with the information given by an encryption with $N = 1$.

It is easy to see that Π_3 is nE secure: If we do not enter in the else, Π_3 is secure. If we enter in the else we observe that c_{l-1} when encrypted with $N = 2$, and c_l when $N = 1$ are independently. We describe formally Π_3 in the extended version [8].

Π_4 a Variant of [9]. Π_4 is obtained from the nE scheme introduced in [9] with these modifications:

- we remove v^* and c_0.
- the if condition becomes if $(N = 1 \vee N = 2) \wedge m_2 = \mathsf{F}_k^{1,0}(1) \oplus m_1$

To enter the if condition during encryption twice, it is necessary to guess $\mathsf{F}_k^{1,0}(1)$ before it is computed. We describe formally Π_4 and the forgery in the extended version [8].

5 Conclusions

We have proved that modes A10, A11, and A12 are not secure in general. This concludes the classification of [20].

Note that our results *do not imply* that all schemes obtained using mode N4, A10, A11, and A12 composition are insecure. Instead, these modes seem insecure only when implemented with artificial schemes, while they are secure when implemented with "natural" schemes. But, these compositions need ad-hoc proofs and cannot rely on general proof.

Finally, this work gives some insights into the limitation of indistinguishability from randomness. That is, having a random ciphertext encrypting the tag may not be enough to make it not usable for forgeries.

Acknowledgements. This work was partly supported by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE. F. Berti was partly funded by the Israel Science Foundation (ISF) grant 2569/21.

References

1. Abed, F., Forler, C., Lucks, S.: General classification of the authenticated encryption schemes for the CAESAR competition. Comput. Sci. Rev. **22**, 13–26 (2016). https://doi.org/10.1016/j.cosrev.2016.07.002
2. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, 19–22 October 1997, pp. 394–403. IEEE Computer Society (1997). https://doi.org/10.1109/SFCS.1997.646128
3. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_41
4. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. J. Cryptol. **21**(4), 469–491 (2008). https://doi.org/10.1007/s00145-008-9026-x

5. Bellare, M., Ng, R., Tackmann, B.: Nonces are noticed: AEAD revisited. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 235–265. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_9
6. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: how to exploit Nonces or redundancy in plaintexts for efficient cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_24
7. Bernstein, D.J.: Caesar call for submissions, final. Technical report (2014). http://competitions.cr.yp.to/caesar.html
8. Berti, F.: Reconsidering generic composition: the modes A10, A11 and A12 are insecure, Cryptology ePrint Archive, Paper 2023/590 (2023). https://eprint.iacr.org/2023/590
9. Berti, F., Pereira, O., Peters, T.: Reconsidering generic composition: the tag-then-encrypt case. In: Chakraborty, D., Iwata, T. (eds.) INDOCRYPT 2018. LNCS, vol. 11356, pp. 70–90. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05378-9_4
10. Berti, F., Pereira, O., Peters, T.: Reconsidering generic composition: the tag-then-encrypt case. In: IACR Cryptol. ePrint Arch., p. 991 (2018). https://eprint.iacr.org/2018/991
11. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Assche, G.V., Keer, R.V.: Farfalle: parallel permutation-based cryptography. IACR Trans. Symmetric Cryptol. **2017**(4), 1–38 (2017). https://tosc.iacr.org/index.php/ToSC/article/view/801
12. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_19
13. Bronchain, O., Momin, C., Peters, T., Standaert, F.: Improved leakage-resistant authenticated encryption based on hardware AES coprocessors. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 641–676 (2021). https://doi.org/10.46586/tches.v2021.i3.641-676
14. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: ASCON v1.2: lightweight authenticated encryption and hashing. J. Cryptol. **34**(3), 33 (2021). https://doi.org/10.1007/s00145-021-09398-9
15. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 15–44. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_2
16. Jimale, M.A., et al.: Authenticated encryption schemes: a systematic review. IEEE Access **10**, 14739–14766 (2022). https://doi.org/10.1109/ACCESS.2022.3147201
17. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, 2nd edn. CRC Press, Boca Raton (2014). https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269
18. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: how secure is SSL?). In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 310–331. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_19
19. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_3

20. Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering generic composition. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 257–274. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_15
21. NIST: Submission requirements and evaluation criteria for the lightweight cryptography standardization process. Technical report (2018). https://csrc.nist.gov/projects/lightweight-cryptography
22. NIST: Lightweight cryptography - finalists. Technical report (2021). http://csrc.nist.gov/Projects/lightweight-cryptography/finalists
23. Peyrin, T., Seurin, Y.: Counter-in-tweak: authenticated encryption modes for tweakable block ciphers. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 33–63. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_2
24. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, 18–22 November 2002, pp. 98–107. ACM (2002). https://doi.org/10.1145/586110.586125
25. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, 6–8 November 2001, pp. 196–205. ACM (2001). https://doi.org/10.1145/501983.502011
26. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_23

Exploring Formal Methods for Cryptographic Hash Function Implementations

Nicky Mouha(✉)

Strativia, Largo, MD, USA
nicky@mouha.be

Abstract. Cryptographic hash functions are used inside many applications that critically rely on their resistance against cryptanalysis attacks and the correctness of their implementations. Nevertheless, vulnerabilities in cryptographic hash function implementations can remain unnoticed for more than a decade, as shown by the recent discovery of a buffer overflow in the implementation of SHA-3 in the eXtended Keccak Code Package (XKCP), impacting Python, PHP, and several other software projects. This paper explains how this buffer overflow vulnerability in XKCP was found. More generally, we explore the application of formal methods to the five finalist submission packages to the NIST SHA-3 competition, allowing us to (re-)discover vulnerabilities in the implementations of Keccak and BLAKE, and also discover a previously undisclosed vulnerability in the implementation of Grøstl. We also show how the same approach rediscovers a vulnerability affecting 11 out of the 12 implemented cryptographic hash functions in Apple's CoreCrypto library. Our approach consists of removing certain lines of code and then using KLEE as a tool to prove functional equivalence. We discuss the advantages and limitations of our approach and hope that our attempt to consolidate some earlier approaches can lead to new insights.

Keywords: SHA-3 · Hash Function · Keccak · BLAKE · Grøstl · CoreCrypto

1 Introduction

A (cryptographic) hash function takes a message of a variable length and turns it into a fixed-length output, known as a "hash value" or "hash." For a hash function to be secure, it should be computationally infeasible to invert the function for a given hash (preimage resistance) or to find two distinct messages that result in the same hash (collision resistance). These properties allow the use of the hash value in place of the message itself in a digital signature scheme, so that successful verification of the signature confirms that the message has not been altered.

In response to the cryptanalysis attack on the SHA-1 hash function presented at CRYPTO 2005 by Wang et al. [36], NIST decided to launch the SHA-3 competition for a new hash function standard [29]. The competition was announced

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 177–195, 2023.
https://doi.org/10.1007/978-3-031-35486-1_9

in November 2007. By October 2008, 64 entries were received, and 51 were selected as first-round candidates in December 2008. Fourteen of these advanced as second-round candidates in July 2009, and the five finalists (BLAKE, Grøstl, JH, Keccak, and Skein) were announced in December 2010. The SHA-3 competition ended in October 2012, when Keccak was declared to be the winner.

Submission packages to the SHA-3 competition were required to include reference and optimized implementations in the C programming language [27]. NIST specified the Application Programming Interface (API) to be used (see Sect. 3) as well as the test vectors that were required in every submission package.

As the submission packages to the SHA-3 competition were subjected to public scrutiny, bugs were reported for several submissions in 2008 and 2009. A systematic analysis by Forsythe and Held of Fortify Software [18] found many bugs that commonly appear in C code, such as out-of-bounds reads, memory leaks, and null pointer dereferences. No bugs were reported during the remainder of the competition, and eventually, the resulting SHA-3 standard became widely implemented in many cryptographic libraries.

However, in September 2015, the implementation on the BLAKE website was updated with the comment: “fixed a bug that gave incorrect hashes in specific use cases” [4]. In 2018, Mouha et al. [24] rediscovered the bug using a new testing methodology that was eventually integrated into Google’s Project Wycheproof [10] and showed that the bug allows the collision resistance of the hash function to be violated.

At CT-RSA 2020, Mouha and Celi [22] showed a vulnerability affecting 11 out of the 12 implemented hash functions in Apple’s CoreCrypto library. The vulnerability required invoking the implementation on inputs of at least 4 GiB, which led to an infinite loop. This vulnerability showed a limitation in NIST’s Cryptographic Algorithm Validation Program (CAVP), which did not perform tests on hash functions for inputs larger than 65 535 bits. To overcome this limitation, NIST introduced the Large Data Test (LDT).

In October 2022, a vulnerability was disclosed by Mouha [31] that impacted both the final-round Keccak submission package and the resulting SHA-3 implementation by its designers. Depending on the specific inputs that were provided, the vulnerability resulted in either an infinite loop or a buffer overflow where attacker-provided values are XORed into memory [23].

Our Contributions. A shortcoming of previous work on finding vulnerabilities in hash function implementations is that they lack generality and clearly fall short as new vulnerabilities keep being found that remained unnoticed for over a decade (in spite of extensive public scrutiny). The novel contribution of this paper is to try to overcome this problem by introducing a new approach that can be used to find vulnerabilities in the hash function implementations of Apple’s CoreCrypto library, as well as in three out of the five SHA-3 finalist submissions: BLAKE, Keccak, and Grøstl. This paper explains how the vulnerability in Keccak was found. The vulnerability in Grøstl is a new contribution in this paper that has not been reported before. Our approach involves symbolic execution to find bugs within seconds, whereas test vectors may take much longer to execute.

2 Background and Related Work

The NIST approach to testing cryptographic implementations dates back to 1977, with the introduction of two sets of test vectors for the Data Encryption Standard (DES) in SP 500-20 [25]. Now known as Known Answer Tests (KATs) or static Algorithmic Functional Tests (AFTs), the first set of test vectors were intended to “fully exercise the non-linear substitution tables” (S-boxes) of the DES. The second set of test vectors, called Monte Carlo Tests (MCTs), contained “pseudorandom data to verify that the device has not been designed just to pass the test set”. Although originally intended to test hardware implementations, this approach can be applied to both hardware and software and forms the basis of NIST’s Cryptographic Algorithm Validation Program (CAVP).

Submissions to the NIST SHA-3 competition were required to include implementations in the C programming language. NIST specified an API [27] and provided source code to generate KATs and MCTs [28]. These KATs and MCTs helped to ensure that various implementations of the same algorithm were consistent. Moreover, an interesting innovation was the inclusion of an Extremely Long Message KAT, which provided a 1 GiB message with the goal of ensuring that large inputs are processed correctly.

For (authenticated) encryption algorithms, the SUPERCOP [7] and BRUTUS [34] frameworks perform some additional tests, such as checking whether overlapping inputs are handled correctly or whether encryption followed by decryption returns the original plaintext.

Aumasson and Romailler introduced crypto differential fuzzing [3] which uses a fuzzer to compare the outputs of different cryptographic libraries and find discrepancies. This approach turned out to be very effective, as shown by the many bugs found by Vranken’s Cryptofuzz project [35].

Formal methods and program verification can also be applied to hash function implementations. Chudnov et al. [15] demonstrated that the Keyed-Hash Message Authentication Code (HMAC) implementation (using the SHA-256 hash function) of Amazon’s s2n library conforms to a formal specification by using Galois’s Software Analysis Workbench (SAW). The HACL* cryptographic library [32,38] is formally verified using the F* verification framework. We refer to Protzenko and Ho [33] for an explanation of how its hash function implementations have recently been completely overhauled. Lastly, we mention Chapman et al.’s SPARKSkein [13] as an implementation of the SHA-3 finalist Skein that was written and verified using the SPARK [1] language and toolset.

Chapman et al. [13] pointed out a bug in the Skein submission package to NIST. The bug involves messages of more than $2^{64} - 8$ bits. Although impractical, this violates a requirement in the SHA-3 call for submissions that a candidate algorithm (and therefore logically also a correct implementation of the algorithm) “shall support a maximum message length of at least $2^{64} - 1$ bits” [26].

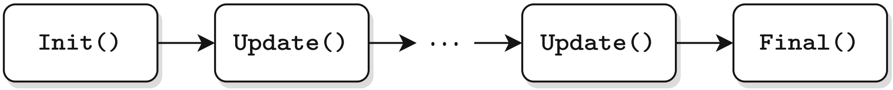


Fig. 1. An evaluation of a hash value on a message that is provided "on the fly" using any number of calls to `Update()` of arbitrary lengths.

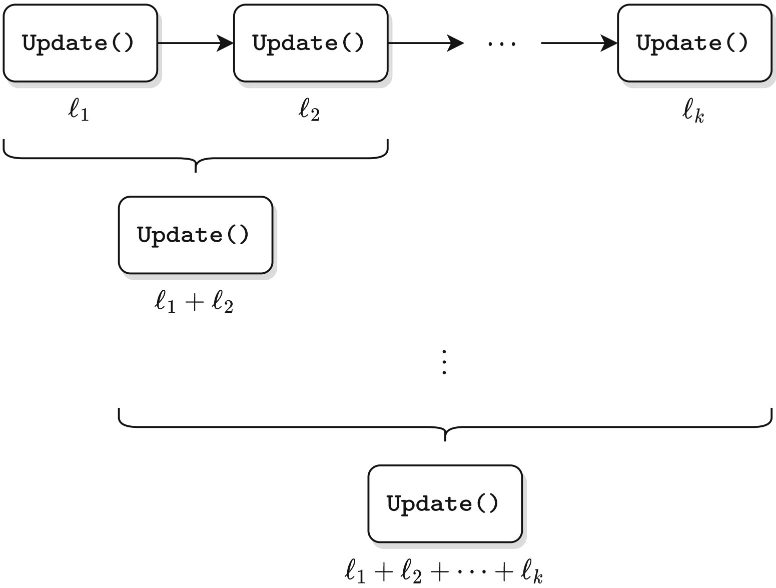


Fig. 2. To prove that any number of calls to `Update()` with arbitrary lengths result in a correct computation, it is sufficient to prove that two calls to `Update()` are equivalent to one larger call to `Update()` on the concatenation of both inputs.

3 Cryptographic Hash Function Interfaces

An API for the SHA-3 competition was specified by NIST [27], requiring the `hashState` data structure and four function calls: `Init()`, `Update()`, `Final()`, and `Hash()`. The API was designed for 64-bit operating systems, which were already common at the start of the SHA-3 competition.

The purpose of the `hashState` data structure is to contain "all information necessary to describe the current state of the SHA-3 candidate algorithm". It must contain the `hashbitlen` variable to indicate the output size of the particular instantiation of the hash function.

The four function calls show how `hashState` is intended to be used:

- `Init()` initializes the `hashState` data structure.
- Once initialized, any number of calls to `Update()` can be made to process parts of the message by updating `hashState` correspondingly. In practice, this "incremental hashing" API offers a major efficiency improvement if the

message is not available all at once or split over two or more non-contiguous arrays [33, p. 9].
- `Final()` performs any final processing needed on `hashState` to output the hash value.
- `Hash()` processes the message all-at-once by calling `Init()`, `Update()`, and `Final()`.

Let us assume that the all-at-once computation using `Init()`, `Update()`, and `Final()` is correct. Then, a sufficient (but not necessary) condition for the correctness of a computation using *any* number of calls to `Update()` (as shown in Fig. 1) is:

Condition 1. *Two consecutive calls to `Update()` change `hashState` in the same way as one call to `Update()` on the concatenation of both inputs.*

It can be seen that Condition 1 is sufficient by means of a proof by contradiction where the condition is applied recursively as illustrated in Fig. 2. However, there are several cases where Condition 1 is not necessary:

- Let us denote a `hashState` as *valid* if and only if is reachable from `Init()` followed by any number of calls to `Update()`. Then, it is not necessary that Condition 1 holds if `hashState` is invalid.
- If the `Final()` function can return the same hash value on two distinct but valid `hashState` data structures, Condition 1 is not necessary either. However, this does not seem to occur in practice, as we have only encountered implementations where `hashState` uniquely represents the message processed so far.
- In the NIST SHA-3 API, all lengths are provided in bits using a 64-bit unsigned integer [27], and a candidate algorithm may impose a maximum message length of $2^{64}-1$ bits [26]. If this maximum message length is imposed, Condition 1 is not necessary for a sequence of two `Update()` calls that exceed the maximum message length (as the hash function may not be defined in this case).
- The NIST SHA-3 API document specifies that all calls to `Update()` contain data lengths (in bits) that are divisible by eight, except possibly the last call. Therefore, for two consecutive calls to `Update()`, we may restrict the first call to a data length that is a multiple of eight bits.

In the next section, our goal will be to verify Condition 1 for a given hash function implementation, possibly along with some preconditions to exclude the aforementioned cases where Condition 1 is not necessary. We will remove some lines of code: as in many previous works we are aiming for the “less ambitious but still important goal of stating partial specifications of program behavior and providing methodologies and tools to check their correctness” [5].

Before concluding this section, note that we will assume throughout this paper that `Init()`, `Update()`, and `Final()` are called in the “correct” order.

In practice it can be desirable to call `Update()` after `Final()`. However, as shown by Benmocha et al. [6], this can be highly insecure for cryptographic libraries that do not expect such usage.

4 Program Verification Using KLEE

In this paper, we will use KLEE [11], which is a symbolic execution tool built on top of the LLVM (Low-Level Virtual Machine) [21] compiler architecture.

Although a typical use of KLEE is to automatically generate test vectors that achieve high code coverage, it can also be used to prove the full functional equivalence of two implementations [11, § 5.5]. Whenever KLEE encounters an execution branch based on a symbolic value, it will (conceptually) follow both branches, maintaining a set of constraints called path conditions for each branch. A downside of this approach is that the number of paths can grow very quickly. To overcome this path explosion problem, KLEE employs a variety of strategies to reduce the number of queries that are sent to the underlying constraint solver.

Unlike CBMC [16], which is a Bounded Model Checker for C and C++ programs, KLEE does not require a bound on the number of iterations for every loop. The NIST SHA-3 competition required a maximum message length of at least $2^{64} - 1$ bits, and it is common to see hash functions that process the message iteratively using a compression function that takes 512 or 1024 bits of input. Computing the hash for such large messages is not possible in practice, however, we will see that our approach using KLEE handles such inputs quite quickly (and without the need for loop unwinding nor manual efforts to rewrite loops).

In the following sections, we will show how to apply KLEE to the reference implementations of the five SHA-3 finalists, as well as to the SHA-3 implementation of XKCP and the hash functions implemented in Apple's CoreCrypto library. In this paper, we only provide the full source code for our KLEE experiments on the JH algorithm. However. the code for all our experiments will be made available as a software artifact.

Table 1 summarizes the runtimes for a 256-bit hash output value, except for Keccak and SHA-3 where we will choose a rate of 1024 bits. We found that the execution time typically does not depend on the length of the hash value. Our experiments were performed on an Intel Core i7-1165G7 processor using KLEE 2.3 with the default STP (Simple Theorem Prover) solver [12,19].

4.1 JH

JH [37] is a hash function designed by Hongjun Wu that advanced to the final round of the NIST SHA-3 competition. As required for all submissions [26], JH supports hash lengths of 224, 256, 384, and 512 bits. The message is padded to a multiple of 512 bits and then split into 512-bit blocks which are processed by the same compression function `F8()`.

Table 1. KLEE runtimes (in minutes and seconds) for a rate of 1024 bits (for Keccak and SHA-3) or a 256-bit hash output length (for all other hash functions). The second column is the runtime to find a test vector that reveals a bug (if the code is incorrect), and the third column is the runtime to prove correctness (after patching the implementation if there is a bug).

Implementation	Buggy	Correct
BLAKE	5 s	11 m 59 s
Grøstl	6 s	10 s
JH	—	50 s
Keccak	1 s	39 s
Skein	—	34 s
XKCP (SHA-3)	1 s	20 s
CoreCrypto	1 s	2 m 41 s

We provide the entire `jh_klee.c` that we used in our experiment in Listing 1. It contains the `Update()` function that is specified in `jh_ref.h` of the JH submission package. For readability, we adjusted the indentation, and for compactness, we removed the source code comments. The only other change that we made to `Update()` is to comment out the lines involving `memcpy()` and `F8()`, as our (partial) equivalence checking does not involve the contents of the message (only its length), nor does it involve the implementation of the compression function `F8()`. Moreover, throughout this paper we do not make any statements about the correctness of `Init()`, `Final()`, nor `Hash()`.

In Listing 2, we provide the Makefile that uses Docker as an easy and portable way to run KLEE on `jh_klee.c`. It will either return a test vector that violates Condition 1, or prove that no such test vector exists. We find that after 50 s, KLEE proves the (partial) consistency of the `Update()` function. An overview of the execution times for all our experiments is given in Table 1.

Five lines in Listing 1 are marked with the comment `// optional` following the reasoning in Sect. 3. They can be safely omitted with the only downside that they roughly double the execution time of KLEE. However, these five lines can be helpful to adapt the approach to other SHA-3 candidate implementations where they may be needed.

Listing 1. Application to JH (`jh_klee.c`).

```
#include <assert.h>
#include <klee/klee.h>

typedef unsigned char BitSequence;
typedef unsigned long long DataLength;
typedef enum { SUCCESS = 0, FAIL = 1,
               BAD_HASHLEN = 2 } HashReturn;
typedef struct {
  int hashbitlen;
  unsigned long long databitlen;
  unsigned long long datasize_in_buffer;
} hashState;

HashReturn Update(hashState *state, const BitSequence
                          *data, DataLength databitlen)
{
  DataLength index;
  state->databitlen += databitlen;
  index = 0;

  if ( (state->datasize_in_buffer > 0 ) &&
       ((state->datasize_in_buffer+databitlen)<512) ) {
    if ( (databitlen & 7) == 0 ) {
      //memcpy(state->buffer +
      //(state->datasize_in_buffer >> 3), data,
      //64-(state->datasize_in_buffer >> 3));
    }
    //else memcpy(state->buffer +
    //          (state->datasize_in_buffer >> 3), data,
    //          64-(state->datasize_in_buffer >> 3)+1);
    state->datasize_in_buffer += databitlen;
    databitlen = 0;
  }

  if ( (state->datasize_in_buffer > 0 ) &&
       ((state->datasize_in_buffer+databitlen)>=512)) {
    //memcpy(state->buffer +
    //       (state->datasize_in_buffer >> 3), data,
    //       64-(state->datasize_in_buffer >> 3) );
    index = 64-(state->datasize_in_buffer >> 3);
    databitlen = databitlen -
                     (512 - state->datasize_in_buffer);
    //F8(state);
    state->datasize_in_buffer = 0;
  }

  for ( ; databitlen >= 512; index = index+64,
                       databitlen = databitlen - 512) {
    //memcpy(state->buffer, data+index, 64);
    //F8(state);
  }

  if (databitlen > 0) {
    //if ((databitlen & 7) == 0)
      //memcpy(state->buffer, data+index,
      //       (databitlen & 0x1ff) >> 3);
    //else
      //memcpy(state->buffer, data+index,
      //       ((databitlen & 0x1ff) >> 3)+1);
    state->datasize_in_buffer = databitlen;
  }

  return(SUCCESS);
}

void test(int hashbitlen) {
  hashState s, s2;
  DataLength databitlen, databitlen1, databitlen2;

  klee_make_symbolic(&s, sizeof(s), "s");
  klee_make_symbolic(&s2, sizeof(s2), "s2");
  klee_make_symbolic(&databitlen, sizeof(databitlen),
                     "databitlen");
  klee_make_symbolic(&databitlen1, sizeof(databitlen1),
                     "databitlen1");
  klee_make_symbolic(&databitlen2, sizeof(databitlen2),
                     "databitlen2");

  s.hashbitlen = hashbitlen;
  s2.hashbitlen = hashbitlen;

  klee_assume(s.databitlen == s2.databitlen);
  klee_assume(s.datasize_in_buffer ==
              s2.datasize_in_buffer);
  klee_assume(s.datasize_in_buffer < 512); // optional
  klee_assume(s2.datasize_in_buffer < 512); // optional

  klee_assume(databitlen == databitlen1 + databitlen2);
  klee_assume(databitlen >= databitlen1); // optional
  klee_assume(databitlen >= databitlen2); // optional
  klee_assume(databitlen1 % 8 == 0); // optional

  Update(&s, NULL, databitlen);

  Update(&s2, NULL, databitlen1);
  Update(&s2, NULL, databitlen2);

  if (s.databitlen != s2.databitlen)
    klee_assert(0);
  if (s.datasize_in_buffer != s2.datasize_in_buffer)
    klee_assert(0);
}

int main() {
  //test(224);
  test(256);
  //test(384);
  //test(512);

  return 0;
}
```

Listing 2. Application to JH (`Makefile` with visible tabs for readability).

```
TARGET = jh_klee

all: $(TARGET)

$(TARGET): $(TARGET).c
␣docker run --rm -v $(CURDIR):/home/klee/host \
␣--ulimit='stack=-1:-1' klee/klee:2.3 \
␣/tmp/llvm-110-install_O_D_A/bin/clang -I \
␣klee_src/include -emit-llvm -c -g3 -O3 \
␣host/$(TARGET).c -o host/$(TARGET).bc
␣time docker run --rm -v $(CURDIR):/home/klee/host \
␣--ulimit='stack=-1:-1' klee/klee:2.3 \
␣klee_build/bin/klee -exit-on-error-type=Assert \
␣host/${TARGET}.bc
␣docker run --rm -v $(CURDIR):/home/klee/host \
␣--ulimit='stack=-1:-1' klee/klee:2.3 \
␣klee_build/bin/ktest-tool $$(docker run --rm -v \
␣$(CURDIR):/home/klee/host --ulimit='stack=-1:-1' \
␣klee/klee:2.3 sh -c "ls \
␣host/klee-last/*.assert.err" | head -n 1 | sed \
␣'s/.assert.err/.ktest/')

clean:
␣\rm -rf *.bc klee-last klee-out-*
```

4.2 Skein

Skein is a final-round SHA-3 submission designed by Ferguson et al. [17]. Like JH, its primary proposal processes the message in 512-bit blocks regardless of the hash value length.

Although the algorithm used by Skein's implementation to process the message in blocks follows a completely different approach compared to JH, the KLEE proving harness looks quite similar with the main difference that the assertion on `datasize_in_buffer` is replaced by an assertion on `u.h.bCnt`.

The execution time is even less than for JH, as KLEE only needs 34 s to prove that there are no test vectors that violate the assertions.

4.3 BLAKE

Another final-round SHA-3 submission is the hash function BLAKE by Aumasson et al. [2]. Depending on the hash length, BLAKE uses either a 512-bit or a 1024-bit compression function. It has a datalen variable to keep track of the number of bits in the buffer, similar to datasize_in_buffer for JH.

However, it also keeps track of a counter for the total number of message bits processed so far. Depending on the hash length, this counter is stored either in an array with two 32-bit unsigned integers, or an array with two 64-bit unsigned integers.

In September 2015, the BLAKE website [4] was updated to correct a bug in all implementations submitted during the SHA-3 competition. Using our approach, KLEE easily rediscovers this bug in just five seconds.

More specifically, for the 256-bit hash value, it provides a test vector showing that Update() on 384 bits followed by 512 bits results in a different state than a single update of 384 + 512 = 896 bits.

This is consistent with the bug conditions described by Mouha et al. [24]: the bug occurs when an incomplete block (less than 512 bits) is followed by a complete block (512 bits).

Using the updated code on the BLAKE website [4], we run into an obstacle when running KLEE. It does not terminate within a reasonable amount of time, as it suffers from the path explosion problem mentioned in Sect. 2.

Further analysis shows that an if-branch inside the while-loop is the culprit of the path explosion. For the 512-bit block size, the BLAKE code is as follows:

```
while( databitlen >= 512 ) {
  state->t32[0] += 512;
  if (state->t32[0] == 0)
    state->t32[1]++;

  //compress32( state, data );
  data += 64;
  databitlen  -= 512;
}
```

We found that this path explosion can be avoided if the counter of the message bits hashed so far is not stored as an array of two unsigned 32-bit variables, but as one unsigned 64-bit variable. More specifically, we change the BLAKE code as follows:

```
while( databitlen >= 512 ) {
  state->t64 += 512;

  //compress32( state, data );
  data += 64;
  databitlen  -= 512;
}
```

With this replacement, KLEE proves that the assertions are unreachable in less than 12 min. Clearly, the execution time is an order of magnitude higher than in the previous examples. It appears that this is due to the additional counter variable used by the BLAKE hash function. If this counter variable is removed, the execution time of KLEE is reduced to only nine seconds.

4.4 Grøstl

We now move on to another SHA-3 finalist: Grøstl by Gauravaram et al. [20]. The message is split into either 512-bit or 1024-bit blocks, depending on the length of the hash value. To the best of our knowledge, no bugs have been reported for this implementation.

In the reference implementation of Grøstl, not all loops are inside `Update()` but also inside the function `Transform()` that does not just process one block, but any number of complete blocks. We already notice a first problem here: all variables representing the message length are 64-bit integers, but the function `Transform()` is declared with a parameter of the (user-defined) 32-bit type `u32`, resulting in an incorrect implicit cast.

As we apply our approach using KLEE, it takes six seconds to find a second bug. The bug is again due to the use of incorrect types: the variable `index` is declared as `int`, which is a 32-bit datatype on 64-bit processors. However, for sufficiently large message inputs, the value of `index` overflows, which results in undefined behavior in the C programming language.

In Listing 3, we provide a program to demonstrate the bug. When compiled using gcc, the program writes a large amount of data into memory, almost certainly resulting in a crash. It gets more interesting when we compile this program using clang. The undefined behavior causes clang to perform an optimization that avoids a buffer overflow but instead outputs the same hash for two messages of a different length. Thereby, the implementation violates the collision resistance property (see Sect. 1).

We searched for implementations that may be vulnerable due to this bug but did not identify any projects or products that might be impacted. For this reason, we did not submit a vulnerability report. The most notable use of Grøstl that we found was as a part of the proof-of-work algorithm of the initial version of the Monero cryptocurrency. However, the use of Grøstl there has long been discontinued.

With the two type errors fixed, proving the correctness using KLEE turned out to be much more difficult than expected. We again face a path explosion problem, which we addressed by hard-coding the block size and rewriting a loop that copied data byte-by-byte into a buffer. With these modifications, KLEE terminated in ten seconds with a proof that the assertions are unreachable.

Listing 3. Due to undefined behavior, the Grøstl bug results in a segmentation fault when compiled using gcc, or a collision when compiled using clang (`groestl_bug.c`).

```
#include <stdio.h>
#include <sys/mman.h>
#include "Groestl-ref.h"

int main() {
  int hashbitlen = 256;
  DataLength len1 = (1uLL<<35) + 8;
  DataLength len2 = 8;

  BitSequence* Msg = (BitSequence*) mmap(NULL, len1/8,
        PROT_READ, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

  BitSequence digest[64];

  printf("Hashing %llu-bit message... \nHash: ", len1);

  Hash(hashbitlen, Msg, len1, digest);

  for (int i = 0; i < hashbitlen/8; i++) {
    printf("%02x", digest[i]);
  }
  printf("\n");

  printf("Hashing %llu-bit message... \nHash: ", len2);

  Hash(hashbitlen, Msg, len2, digest);

  for (int i = 0; i < hashbitlen/8; i++) {
    printf("%02x", digest[i]);
  }
  printf("\n");

  return 0;
}
```

4.5 Keccak

The only SHA-3 finalist that we did not yet study in this paper, is the submission that won the competition: Keccak by Bertoni et al. [9]. Keccak pads the message and splits it into blocks, which are then processed by a cryptographic permutation. The block size, also known as the rate, has a default value of 1024 bits [9]. We will focus on this default value for now, and discuss the impact of the block size later.

A vulnerability was reported by Mouha and assigned CVE-2022-37454 [31]. As the winner of the SHA-3 competition, the Keccak reference code is quite widespread, and the vulnerability impacted various projects such as Python and PHP. For the details of the vulnerability, we refer to Mouha and Celi [23]. In this paper, we explain how the vulnerability was discovered using KLEE.

A straightforward approach using KLEE does not terminate in a reasonable amount of time. Therefore, it can be good to rule out an infinite loop in the Keccak implementation, as this would lead KLEE to enter into an infinite loop as well.

For `while(i < databitlen)` to terminate, a sufficient but not necessary condition is that `i` advances in every loop iteration. This is easy to check by introducing an `old_i` variable that is initialized to `i`. When the variable `i` is modified, we ensure that it is different from the previous loop iteration:

```
if (i == old_i) klee_assert(0);
```

At the end of the loop, we set `old_i = i`. With this additional code to detect infinite loops, it takes KLEE less than a second to output an assertion error. By analyzing the test vector provided by KLEE, we can confirm that we have indeed found an infinite loop. Note that this additional code is only used to allow us to easily detect an infinite loop in KLEE, the additional code is not necessary for a correct implementation (which does not contain an infinite loop).

It turns out that there is not only an input that leads to an infinite loop, but another input that causes a large amount of data to be written into memory, leading to a segmentation fault. The details of this buffer overflow are given by Mouha and Celi [23] and outside the scope of this paper.

The bug presents itself when there are already x bits of data in the buffer, and then an `Update()` of $2^{32} - x$ bits or more is made. Two problems can occur in Keccak's code below: the higher bits may be discarded due to an incorrect cast to a 32-bit integer, and the addition may overflow:

```
partialBlock = (unsigned int) (databitlen - i);
if (partialBlock + state->bitsInQueue > state->rate)
  partialBlock = state->rate - state->bitsInQueue;
```

We can correct this bug by rearranging the code a bit, so that `partialBlock` is at most equal to the block size. In that case, a 32-bit integer suffices for `partialBlock`:

```
if (databitlen - i > state->rate - state->bitsInQueue)
  partialBlock = state->rate - state->bitsInQueue;
else
  partialBlock = (unsigned int) (databitlen - i);
```

For the corrected code, KLEE requires only 39 s to prove that the assertions cannot be violated.

Finally, we note that KLEE did not seem to terminate within a reasonable amount of time for block sizes that are not a multiple of two. Such block sizes were proposed during the SHA-3 competition to handle different levels of security. Unfortunately, it appears to be a common issue that solvers have difficulties handling divisions by a constant that is not a power of two. KLEE has a `-solver-optimize-divides` flag that tries to optimize such divisions before passing them to the solver. However, even with this flag, we could not find a way to make KLEE terminate for block sizes that are not a multiple of two.

4.6 XKCP (SHA-3)

The eXtended Keccak Code Package (XKCP) [8] is maintained by the Keccak team. It contains the NIST-standardized variant of the Keccak hash function. Between the final-round Keccak submission and the SHA-3 standard, only the message padding is different.

The implementation of SHA-3 in XKCP is based on the implementation of the final-round Keccak submission. However, it has gone through quite a bit of refactoring. For this reason, we list XKCP's SHA-3 as a separate implementation in Table 1.

The same bug that impacts the final-round Keccak submission is present in XKCP as well, although two calls to `Update()` with a combined length of 2^{32} bytes (4 GiB) rather than 2^{32} bits (512 MiB) are required to trigger it. Moreover, it has another bug (not present in the Keccak submission) where messages slightly below 2^{64} bytes result in an infinite loop. This is due to an overflow in the comparison operation (`dataByteLen >= (i + rateInBytes)`), which has been replaced by `dataByteLen-i >= rateInBytes` in the corrected version of the code.

Using KLEE, it takes less than one second to provide a test vector that triggers the bug, assuming we again augment the code to detect infinite loops. For a 1024-bit block size, KLEE requires 20 s to prove that the assertions are unreachable.

4.7 CoreCrypto

Lastly, we would like to revisit the infinite loop in Apple's CoreCrypto library. The vulnerability impacted 11 out of the 12 implemented hash functions and was assigned CVE-2019-8741 [30]. For details of the bug, we refer to Mouha and Celi [22].

The approach using KLEE is quite straightforward to implement. KLEE finds a vulnerable test vector in less than a second for the original implementation and can prove that the assertions are unreachable in less than three minutes for the updated implementation.

We want to point out an interesting coincidence here. If we start from Apple CoreCrypto's corrected implementation of `Update()`, with just a little bit of refactoring (such as replacing `len` by `dataByteLen-i`, renaming variables and functions, and removing unneeded code), we end up with the corrected implementation of `Update()` that is used by XKCP. It seems that `Update()` is simple enough that two teams can independently arrive at the same implementation (up to simple refactoring), but complex enough to contain vulnerabilities that remained undiscovered for over a decade.

5 Limitations and Discussion

After the SHA-3 competition ended in 2012, vulnerabilities were found in the implementations of the SHA-3 finalist BLAKE in 2015 [4], in 11 out of the 12 implemented hash functions of Apple's CoreCrypto library in 2019 [30], and recently in the reference implementation of the SHA-3 winner Keccak [31].

These vulnerabilities were all related to the `Update()` function that is used to process the message in blocks. Nevertheless, the impact of the vulnerabilities is quite different. Whereas XKCP's SHA-3 implementation contained a buffer overflow vulnerability with the possibility of arbitrary code execution, the impact of the vulnerability in Apple's CoreCrypto library is limited to an infinite loop. The BLAKE vulnerability cannot be used to trigger any runtime error, however, it can be used to violate the collision resistance property of the hash function.

This allows us to make a first observation that approaches to avoid memory safety problems (such as enforcing coding standards, sandboxing, or moving away from C/C++ to safer programming languages) would be helpful, but not sufficient to avoid the vulnerabilities described in this paper. We are also reaching the limits of approaches using test vectors: the Large Data Test (LDT) can take quite a long time to execute on a 4 GiB message to detect bugs in Apple's CoreCrypto library, and for the XKCP bug, a single large call to `Update()` does not trigger the vulnerability (as it requires that some data is already present in the buffer).

Therefore, it can be interesting to consider approaches using symbolic execution. The approach we describe in this paper using KLEE turns out to be quite easy to deploy. We performed (partial) equivalence checking on the `Update()` function with lines involving the message contents and the compression function commented out. In a production environment, the proving harness would override these functions rather than commenting them out, so that the proofs can be part of a continuous integration process similar to how Amazon Web Services currently deploys CBMC [14].

Whereas approaches using large test vectors can take quite some time to execute (especially on slow hardware), symbolic execution can find bugs in a few seconds or prove correctness in less than 12 min, as shown by our timings in Table 1. This makes our approach a low-cost entry towards formal methods and program verification, and perhaps even a stepstone towards more rigorous approaches used in projects such as HACL* [32,38] or SPARKSkein [13].

Indeed, the litmus test here would be to see how easily our approach extends to other submissions in the SHA-3 competition. We studied the implementations of all five finalists of the competition and found that we can either prove correctness or unearth new bugs (as in the case of Grøstl, where we show how undefined behavior can violate the collision resistance of the hash function implementation). And although our approach intends to check whether two calls to `Update()` are consistent with one call on the concatenation of both inputs, it also finds bugs that can be triggered by one call to `Update()`, as shown by the bugs in Grøstl and Apple's CoreCrypto library (as they cause KLEE to enter into an infinite loop).

Lastly, although our approach was helpful to discover bugs, we stress again that it is insufficient to claim that the implementations are correct. For example, we make no statements about potential bugs in `Init()`, `Final()`, or `Hash()`, nor potential bugs in the lines of `Update()` that were commented out. We also do not look into bugs related to the use of the API, such as those found by Benmocha et al. [6].

6 Conclusion and Future Work

We revisited the implementations of the five finalists of the NIST SHA-3 competition. These had been subject to a rigorous public review process from 2008 to 2012. However, it was not until 2015 that a vulnerability was discovered in the implementation of BLAKE, and very recently in the winning Kecak submission using the technique that is first described in this paper.

We showed how these bugs can be (re-)discovered in only a matter of seconds, requiring only minimal effort to construct a proving harness for the original code. Moreover, we also found a vulnerability in the Grøstl submission, allowing the construction of two messages that result in the same hash value when compiled using clang. Our approach would also have discovered a bug in Apple's CoreCrypto library that was reported in 2019.

Our approach requires symbolic execution to check whether two `Update()` calls (with some lines of code removed) are equivalent to one `Update()` call on the concatenation of both inputs. To check this property, we used the KLEE symbolic execution framework.

Unfortunately, our approach involves a bit of trial and error. In particular, to prove that none of the assertions fail, we sometimes needed to rewrite the code a bit to avoid that KLEE fails to terminate due to path explosion. This is a limitation as we would ideally like to make no changes at all to the source code, but perhaps this is an acceptable compromise to achieve the goal of (partial) program verification.

An interesting direction for future work is to find a way to extend our approach to Keccak and SHA-3 when the block size is not a power of two.

References

1. AdaCore, Thales: Implementation Guidance for the Adoption of SPARK (2020). https://www.adacore.com/uploads/books/pdf/Spark-Guidance-1.2-web.pdf
2. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal BLAKE. submission to the NIST SHA-3 competition (round 3) (2010). https://www.aumasson.jp/blake/blake.pdf
3. Aumasson, J.P., Romailler, Y.: Automated testing of crypto software using differential fuzzing. Black Hat USA 2017 (2017). https://yolan.romailler.ch/ddl/talks/CDF-wp_BHUSA2017.pdf
4. Aumasson, J.P.: SHA-3 proposal BLAKE (2015). https://web.archive.org/web/20150921185010/https://131002.net/blake/
5. Ball, T., Cook, B., Levin, V., Rajamani, S.K.: SLAM and static driver verifier: technology transfer of formal methods inside Microsoft. In: Boiten, E.A., Derrick, J., Smith, G. (eds.) IFM 2004. LNCS, vol. 2999, pp. 1–20. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24756-2_1

6. Benmocha, G., Biham, E., Perle, S.: Unintended features of APIs: cryptanalysis of incremental HMAC. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 301–325. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81652-0_12
7. Bernstein, D.J., Lange, T.: eBACS: ECRYPT benchmarking of cryptographic systems (2022). https://bench.cr.yp.to
8. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Assche, G.V., Keer, R.V.: eXtended Keccak code package (2022). https://github.com/XKCP/XKCP
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak SHA-3 submission. Submission to the NIST SHA-3 competition (round 3) (2011). https://keccak.team/files/Keccak-submission-3.pdf
10. Bleichenbacher, D., Duong, T., Kasper, E., Nguyen, Q.: Project Wycheproof (2019). https://github.com/google/wycheproof
11. Cadar, C., Dunbar, D., Engler, D.R.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Draves, R., van Renesse, R. (eds.) OSDI 2008, pp. 209–224. USENIX Association (2008)
12. Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R.: EXE: automatically generating inputs of death. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) CCS 2006, pp. 322–335. ACM (2006). https://doi.org/10.1145/1180405.1180445
13. Chapman, R., Botcazou, E., Wallenburg, A.: SPARKSkein: a formal and fast reference implementation of skein. In: Simao, A., Morgan, C. (eds.) SBMF 2011. LNCS, vol. 7021, pp. 16–27. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25032-3_2
14. Chong, N., et al.: Code-level model checking in the software development workflow at Amazon web services. Softw. Pract. Exp. **51**(4), 772–797 (2021). https://doi.org/10.1002/spe.2949
15. Chudnov, A., et al.: Continuous formal verification of Amazon s2n. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10982, pp. 430–446. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96142-2_26
16. Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24730-2_15
17. Ferguson, N., et al.: The skein hash function family. Submission to the NIST SHA-3 competition (round 3) (2010). https://www.schneier.com/wp-content/uploads/2015/01/skein.pdf
18. Forsythe, J., Held, D.: NIST SHA-3 competition security audit results (2009). https://web.archive.org/web/20120222155656if_/http://blog.fortify.com/repo/Fortify-SHA-3-Report.pdf
19. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 519–531. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_52
20. Gauravaram, P., et al.: Grøstl - a SHA-3 candidate. Submission to the NIST SHA-3 competition (round 3) (2011). https://www.groestl.info/Groestl.pdf
21. Lattner, C., Adve, V.S.: LLVM: a compilation framework for lifelong program analysis & transformation. In: CGO 2004, pp. 75–88. IEEE Computer Society (2004). https://doi.org/10.1109/CGO.2004.1281665
22. Mouha, N., Celi, C.: Extending NIST's CAVP testing of cryptographic hash function implementations. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 129–145. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40186-3_7

23. Mouha, N., Celi, C.: A vulnerability in implementations of SHA-3, SHAKE, EdDSA, and other NIST-approved algorithms. In: Rosulek, M. (ed.) CT-RSA 2023. LNCS, vol. 13871, pp. 3–28. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30872-7_1
24. Mouha, N., Raunak, M.S., Kuhn, D.R., Kacker, R.: Finding bugs in cryptographic hash function implementations. IEEE Trans. Reliab. **67**(3), 870–884 (2018). https://doi.org/10.1109/TR.2018.2847247
25. National Bureau of Standards: Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard. NBS Special Publication 500-20 (1977). https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nbsspecialpublication500-20e1977.pdf
26. National Institute of Standards and Technology: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. 72 Fed. Reg. (2007). https://www.federalregister.gov/d/E7-21581
27. National Institute of Standards and Technology: ANSI C Cryptographic API Profile for SHA-3 Candidate Algorithm Submissions (2008). https://csrc.nist.gov/CSRC/media/Projects/Hash-Functions/documents/SHA3-C-API.pdf
28. National Institute of Standards and Technology: Description of Known Answer Test (KAT) and Monte Carlo Test (MCT) for SHA-3 Candidate Algorithm Submissions (2008). https://csrc.nist.gov/CSRC/media/Projects/Hash-Functions/documents/SHA3-KATMCT1.pdf
29. National Institute of Standards and Technology: Hash Functions: SHA-3 Project (2020). https://csrc.nist.gov/projects/hash-functions/sha-3-project
30. National Vulnerability Database: CVE-2019-8741 (2020). https://nvd.nist.gov/vuln/detail/CVE-2019-8741
31. National Vulnerability Database: CVE-2022-37454 (2022). https://nvd.nist.gov/vuln/detail/CVE-2022-37454
32. Polubelova, M., et al.: HACLxN: verified generic SIMD crypto (for all your favourite platforms). In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS 2020, pp. 899–918. ACM (2020). https://doi.org/10.1145/3372297.3423352
33. Protzenko, J., Ho, S.: Functional pearl: zero-cost, meta-programmed, dependently-typed stateful functors in F*. CoRR abs/2102.01644 (2021). https://arxiv.org/abs/2102.01644
34. Saarinen, M.J.O.: BRUTUS (2016). https://github.com/mjosaarinen/brutus
35. Vranken, G.: Cryptofuzz - differential cryptography fuzzing (2022). https://github.com/guidovranken/cryptofuzz
36. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_2
37. Wu, H.: The hash function JH. Submission to the NIST SHA-3 competition (round 3) (2011). https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf
38. Zinzindohoué, J.K., Bhargavan, K., Protzenko, J., Beurdouche, B.: HACL*: a verified modern cryptographic library. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) CCS 2017, pp. 1789–1806. ACM (2017). https://doi.org/10.1145/3133956.3134043

Public-Key Cryptography

A Tightly Secure ID-Based Signature Scheme Under DL Assumption in AGM

Jia-Chng Loh[1](✉), Fuchun Guo[1], Willy Susilo[1], and Guomin Yang[2]

[1] Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, Australia
{jial,fuchun,wsusilo}@uow.edu.au

[2] Singapore Management University, Singapore, Singapore
gmyang@smu.edu.sg

Abstract. Identity-based signatures (IBS) can be verified using the signer identity information as the public key, and hence, there is no need for certificate management that proves the corresponding public key ownership. Unfortunately, none of the existing IBS schemes has security proven as tight as the discrete logarithm (DL) problem, *the hardest problem* in the cyclic group setting, under the standard EUF-CMA security model. Recently, the introduction of proving security in the algebraic group model (AGM), where the adversary's computation is algebraic, enables some ordinary signature schemes to be proven tightly reducible under DL assumption and EUF-CMA. To date, however, it remains unknown whether IBS schemes can also be proven as secure as the DL problem in the AGM. Achieving tight security in IBS schemes under standard EUF-CMA is challenging, due to the need to take extra precautions against adaptive queries on user private keys by the adversary. In this work, we show, for the first time, an IBS scheme with tight security under DL assumption and EUF-CMA in the AGM. The scheme features a minimal signature size of two group elements, with a reduction loss factor of two.

Keywords: Identity-based signatures · Provable security · Tight reduction · Algebraic group model

1 Introduction

Identity-Based Signature with the Ideal Security. Digital signature is one of the main cryptographic primitives to enable authenticated electronic communications [15]. However, the public key infrastructure (PKI) is necessary to provide a certificate verifying the validity of the signer's public key in practical use. The notion of identity-based signatures (IBS) was introduced by Shamir [37], where the signer's identity, such as an ID number or email address, serves as the public key, eliminating the need for a certificate verifying the signer's authority. Therefore, IBS can be publicly verified via the signer's identity. The standard security model for IBS, namely existential unforgeability against chosen identity-and-message attacks (EUF-CMA) [5], guarantees that it's impossible to forge a

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 199–219, 2023.
https://doi.org/10.1007/978-3-031-35486-1_10

new signature on a unique message and identity pair, even if the attacker has access to many user private keys and many signatures.

When proving the security of a cryptographic scheme, the security reduction is tight if security loss L is a constant or a small number $L \geq 1$. Security loss is an important factor when it comes to deciding the concrete size of the security parameter of a scheme in practice because we must increase the size of the security parameter so that it compensates for the security loss, unfortunately, it degrades the scheme efficiency. On the other hand, the security of a scheme is determined by the difficulty of the underlying problem. The harder the problem, the stronger the security provided by the scheme. Considering the cyclic groups setting, it is known that the discrete logarithm (DL) problem is the hardest one because algorithms that can solve the DL problem can be used to solve the computational Diffie-Hellman (CDH) problem and its variants [22,28]. Therefore, schemes with ideal security, i.e., as secure as the DL problem in the standard EUF-CMA security model, offer the strongest security assurance in the context of cyclic groups.

There are several IBS schemes that have been proven secure under the EUF-CMA security model and DL assumption [5,12,20,31] or CDH assumption and its variants [3,13,23,27,33,34]. However, none of these schemes achieve the ideal security, meaning their security cannot be tightly reduced to the DL problem under EUF-CMA.

The Algebraic Group Model. The concept of the algebraic algorithm was initially introduced by Boneh and Venkatesan in [9]. It has since been employed in several studies, including those that aim to demonstrate impossibilities [1,9,10,14,21,26,32]. The formalization of the algebraic group model (AGM) was later carried out by Fuchsbauer, Kiltz, and Loss in [16]. In the AGM, the security of a scheme is proven through a security reduction that demonstrates how a probabilistic polynomial-time (PPT) adversary can compromise the scheme, by idealizing the adversary's computations as algebraic, yielding representations that describe how the output group element can be generated from received group elements.

In the EUF-CMA security model, due to the algebraic nature of the adversary's forgeries and representations, some standard signature schemes such as BLS [8] and Schnorr [36] can be reduced in the AGM with random oracles to extract DL solutions with an overwhelming success probability. This has been shown in works such as [16,17], respectively. On the other hand, variants of secure signature schemes have been analyzed in the AGM [2,4,24,25,30]. However, to date, the security of identity-based signatures (IBS) in the AGM has not yet been studied. Hence, it is unknown whether the security of IBS schemes can also be proven as tight as the DL problem in the AGM under EUF-CMA.

Challenge. It is not a trivial task to have tight security for IBS schemes, especially in the standard EUF-CMA security model, due to the adversary may adaptively ask for user private keys during the query phase. Therefore, a major challenge in achieving tight reduction for IBS under the DL assumption and EUF-CMA is developing a reduction algorithm that can simultaneously (1) respond to user private key queries and (2) extract problem solutions based on forged signatures. However, these two tasks are often conflicting, if the user private key is simulated, the forged signature is often not reducible in security reductions.

Existing reduction techniques [11,13,20,27,34] address the first challenge by dividing the simulator's handling of user identity and signature queries, resulting in at least one target forgery with a non-simulatable user private key. If the chosen identity of the forgery matches the target, the simulator can extract the problem solution during the forgery phase, bypassing the second challenge. However, this approach leads to a loose reduction due to the random selection of the target identity and a security loss of at least q query times.

1.1 Contribution

In this work, we show, for the first time, how to obtain a tightly secure IBS scheme under DL assumption and EUF-CMA by adopting the algebraic adversary. The contributions are summarized as follows.

We first present a new IBS scheme, named BLS-IBS, which is extended from the ordinary BLS signature scheme [8] to the identity-based setting. This results in signatures consisting of only two group elements. For example, let $(p, g, \mathbb{G})$ be a bilinear group tuple in prime order p with a generator $g \in \mathbb{G}$ and $\mathcal{H}_1, \mathcal{H}_2 :\rightarrow \mathbb{G}$ be cryptographic hash functions. The signature $\sigma_{ID,m} = (\sigma_{ID,m}^{(1)}, \sigma_{ID,m}^{(2)})$ on identity and message pair (ID, m) is defined as

$$\sigma_{ID,m}^{(1)} = d_{ID} \cdot \mathcal{H}_2(m)^r, \quad \sigma_{ID,m}^{(2)} = g^r,$$

where $d_{ID} = \mathcal{H}_1(ID)^x$ is the user private key that can be obtained based on the BLS signatures with master secret key $x \in \mathbb{Z}_p$.

To demonstrate that the security of the BLS-IBS scheme can be tightly reduced to the DL problem, we propose a security reduction in the AGM that addresses the two aforementioned challenges. Specifically,

- Our reduction can simulate any user private key, which enables the simulator to respond to all user private key queries and signature queries without aborting during the query phase of the EUF-CMA security game, regardless of the identities that have been queried for signatures.
- It is possible to reduce any forgery, even if the private key of the forged identity is simulatable. The algebraic adversary's forgery and representations play a crucial role in this reduction process, helping the simulator find a solution to the problem.

It is also worth noting that the security analysis in the AGM of BLS-IBS is more complex than that of ordinary BLS in the AGM due to the fact that the adversary's forgery and representations can be classified into several cases, rather than just two cases. In particular, we design two indistinguishable simulations in the AGM with random oracles that can simulate any user private key and extract DL solution with at least half of the success probability, resulting in a tight reduction with a loss factor of two.

1.2 Other Related Work

IBS Schemes Under DL Assumption and EUF-CMA. The first secure EUF-CMA IBS scheme under DL assumption, known as Beth-IBS, was derived from Bellare et al.'s framework [5] transformed Beth's identification scheme [7] to IBS. However, due to the lack of security analysis for the Beth-IBS scheme, Galindo and Garcia [20] proposed a new IBS scheme based on the well-known Schnorr signature [36] in the random oracle model [6], namely Schnorr-like IBS scheme. The Schnorr-like IBS scheme is considered the most efficient IBS to date, due to its pairing-free setting. Although its security was improved by Chatterjee et al. [11], it is still loosely reduced to the DL problem due to the need for the reset lemma [35], which has been proven that the resulting security loss must suffer from the tightness barrier – the reduction loss cannot be tight [26].

IBS Schemes Under CDH Assumption. The first IBS scheme without random oracles was proposed by Paterson and Schuldt [34], based on Waters' signatures [38], and secure under the EUF-CMA and CDH assumption. Narayan and Parampalli [29] further improved the scheme by reducing the size of the public parameters. Unfortunately, their security reductions are not tight and the computation cost is high. Recently, Yi et al. [39] proposed a new IBS scheme with a more efficient computation cost, but the reduction loss remains non-tight under the CDH assumption.

Tightly Secure IBS Schemes Under the Strong Assumption and Weak Security Model. Recently, Fukumitsu and Hasegawa [18,19] proposed enhancements to the Galindo and Garcia's Schnorr-like IBS [20] to design IBS schemes with a tight security reduction. However, these two schemes are only proven under a strong assumption, i.e., the decisional Diffie-Hellman assumption, and in the weak-EUF-CMA security model [40], where the adversary is restricted to asking for a user's private key if the identity has already been requested for signatures.

2 Preliminaries

2.1 Definition of Identity-Based Signatures

An identity-based signature (IBS) scheme consists of the following algorithms.

- *Setup*(1^k): A setup algorithm that generates the master public and secret key pair (mpk, msk) when given the security parameters 1^k as input.
- *Extract*(mpk, msk, ID): A user private key extraction algorithm that takes as input the master public and secret key pair (mpk, msk) and a user identity ID, and returns the user private key d_{ID}.
- *Sign*(mpk, d_{ID}, m): An algorithm for generating a signature, which takes as input the master public key mpk, user's private key d_{ID} and message m, and outputs a signature $\sigma_{ID,m}$.
- *Verify*$(mpk, ID, m, \sigma_{ID,m})$: An algorithm for verifying inputs $(mpk, ID, m, \sigma_{ID,m})$ which outputs either 1 for acceptance or 0 for rejection.

Correctness. For any user identity ID and user private key d_{ID} that is generated based on the master public and secret key pair (mpk, msk), i.e., $d_{ID} \leftarrow Extract(mpk, msk, ID)$, signing a message m with d_{ID} must return a correct signature $\sigma_{ID,m} \leftarrow Sign(mpk, d_{ID}, m)$ that is valid on user ID, i.e., the verification holds $1 \leftarrow Verify(mpk, ID, m, \sigma_{ID,m})$.

2.2 Security of EUF-CMA for IBS

The notion of existential unforgeability against chosen identity-and-message attacks (EUF-CMA) [5] security model for IBS schemes is defined between a challenger and an adversary as follows.

- **Setup Phase:** Let L_E be a set of extraction queries. The challenger runs *Setup* algorithm to compute a master public and secret key pair (mpk, msk). The challenger forwards mpk to the adversary.
- **Query Phase:** The adversary adaptively asks for user private keys d_{ID_i} on any identity ID_i to the extraction oracle $\mathcal{O}_E$ and signatures σ_{ID_i,m_i} on any identity and message pair (ID_i, m_i) to the signing oracle $\mathcal{O}_S$.

  Extraction oracle $\mathcal{O}_E(ID_i)$: On input i-th query for ID_i, it first checks whether $\langle ID_i, \cdot\rangle \in L_E$ exists. The challenger retrieves user private key d_{ID_i} if it finds $\langle ID_i, d_{ID_i}\rangle$. Otherwise, the challenger calls $Extract(mpk, msk, ID_i) \rightarrow d_{ID_i}$ algorithm and stores $\langle ID_i, d_{ID_i}\rangle$ to L_E. The challenger returns d_{ID_i}.

  Signing oracle $\mathcal{O}_S(ID_i, m_i)$: On input i-th query for (ID_i, m_i), it first checks and retrieves d_{ID_i} if $\langle ID_i, d_{ID_i}\rangle \in L_E$ exists. If $(ID_i, d_{ID_i}) \notin L_E$, the challenger calls extraction algorithm $Extract(mpk, msk, ID_i) \rightarrow d_{ID_i}$ and stores user private key $\langle ID_i, d_{ID_i}\rangle$ to L_E. At the end, the challenger calls signing algorithm $Sign(mpk, d_{ID_i}, m_i) \rightarrow \sigma_{ID_i,m_i}$. The challenger returns σ_{ID_i,m_i}.
- **Forgery Phase:** The adversary returns a forged signature σ_{ID^*,m^*} on a chosen identity and message pair (ID^*, m^*). The adversary wins if the verification holds $Verify(mpk, ID^*, m^*, \sigma_{ID^*,m^*}) = 1$, such that ID^* has not been queried to $\mathcal{O}_E$ and (ID^*, m^*) has not been queried to $\mathcal{O}_S$.

Definition 1. *An IBS scheme is (ϵ, q_e, q_s, t)-secure in the EUF-CMA security model if it is infeasible that any probabilistic polynomial-time adversary who runs in t polynomial time and makes at most q_e extraction queries and q_s signing queries has an advantage at most ϵ in winning the game, where ϵ is a negligible function of the input security parameter.*

2.3 Bilinear Discrete Logarithm Problem

Let $\mathbb{G}, \mathbb{G}_T$ be groups of prime order p with generator $g \in \mathbb{G}$. A bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is defined in the pairing group setting. The discrete logarithm (DL) problem is to compute $a \in \mathbb{Z}_p$, given a random group element $g^a \in \mathbb{G}$.

Definition 2. *The (ϵ, t)-DL assumption holds in $\mathbb{G}$ if there is no probabilistic polynomial-time adversary who runs in t polynomial time has an advantage at most ϵ to solve the DL problem in $\mathbb{G}$.*

2.4 The Algebraic Algorithms

Fuchsbauer, Kiltz, and Loss [16] formalized the algebraic group model (AGM), which considers the adversary's computation as algebraic. The definition of the algebraic algorithm is described as follows.

Definition 3. *(Algebraic Algorithm, Definition 2.1 of [16]) Suppose the adversary in the security game is algebraic, and it is given* $(X_0, X_1, ..., X_n) \in \mathbb{G}^{n+1}$ *group elements, such that* $X_0 = g \in \mathbb{G}$ *be the group generator. Whenever the adversary outputs* $Z \in \mathbb{G}$*, it also attaches a representation vector* $[\vec{\pi}] \in \mathbb{Z}_p^{n+1}$*, where* $[\vec{\pi}] = (\pi_0, \pi_1, ..., \pi_n) \in \mathbb{Z}_p^{n+1}$*, that indicates how* Z *is generated based on received group elements, such that* $Z = \prod_{i=0}^{n} X_i^{\pi_i}$*.*

3 The BLS-IBS Scheme

The BLS-IBS scheme is extended based on the BLS signature [8] to the identity-based setting. The scheme's algorithms are defined as follows.

- *Setup*: On input security parameter 1^k. Let $\mathbb{G}$ be the notion of groups, $g \in \mathbb{G}$ be the generator of group, p be the prime order, $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a bilinear map, where $\mathbb{G}_T$ denotes the *target group*, and $\mathcal{H}_1, \mathcal{H}_2 : \{0,1\}^* \rightarrow \mathbb{G}$ be two cryptography hash functions. It selects $x \in \mathbb{Z}_p^*$ and computes $X = g^x$. The master public and secret key pair (mpk, msk) is returned as
$$mpk = (g, p, e, \mathbb{G}, \mathbb{G}_T, \mathcal{H}_1, \mathcal{H}_2, X), \quad msk = x.$$
- *Extract*: On input (mpk, msk) and user identity $ID \in \{0,1\}^*$. It calls $\mathcal{H}_1(ID) = H_{ID} \in \mathbb{G}$ and computes user private key d_{ID} as
$$d_{ID} = (H_{ID})^x.$$
- *Sign*: On input (mpk, d_{ID}, m). It calls $\mathcal{H}_2(m) = H_m \in \mathbb{G}$ and randomly selects $r \in \mathbb{Z}_p^*$. The signature $\sigma_{ID,m} = (\sigma_{ID,m}^{(1)}, \sigma_{ID,m}^{(2)})$ on (ID, m) is returned as
$$\sigma_{ID,m}^{(1)} = d_{ID} \cdot (H_m)^r,$$
$$\sigma_{ID,m}^{(2)} = g^r.$$
- $Verify$: On input $(mpk, ID, m, \sigma_{ID,m})$, it checks whether or not the following equation holds
$$e(\sigma_{ID,m}^{(1)}, g) = e(H_{ID}, X) \cdot e(H_m, \sigma_{ID,m}^{(2)}).$$

Correctness. Recall that $X = g^x$ is part of mpk. The scheme is correct if the validity of signature $\sigma_{ID,m} = (\sigma_{ID,m}^{(1)}, \sigma_{ID,m}^{(2)})$ on identity and message pair (ID, m) generated with user private key d_{ID} holds.

$$\begin{aligned} e(\sigma_{ID,m}^{(1)}, g) &= e(H_{ID}, X) \cdot e(H_m, \sigma_{ID,m}^{(2)}) \\ &= e(\mathcal{H}_1(ID)^x \cdot \mathcal{H}_2(m)^r, g). \end{aligned}$$

4 Security Proof

In this section, we begin by presenting the high-level security proof for the BLS-IBS scheme in the AGM with random oracles under EUF-CMA and DL assumption. Then, we detail how we design two indistinguishable simulations that can simulate any user private key. Afterwards, we classify the algebraic adversary's forgery and representations into several cases and show that the simulator can extract for problem solution in any of them. Finally, we provide full security proof.

High-Level. Given a DL problem instance tuple (g, g^a), we design two indistinguishable simulations $\mathcal{R}_1, \mathcal{R}_2$, which control hash responses as the random oracles. We set simulation $\mathcal{R}_1$ to embed g^a into master public key X and every signature randomness $\sigma^{(2)}_{ID_i,m_i}$; or simulation $\mathcal{R}_2$ to embed g^a into every hash responses H_{ID_i}, H_{m_i} – including H_{ID^*}, H_{m^*}. The simulation setup is detailed as in Table 1, where all $x, h_{ID_i}, h_{m_i}, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, s_i, t_i, r_i \in \mathbb{Z}_p^*$ are randomly chosen. During the forgery phase, we classify the algebraic adversary's forgery and representations into several cases as illustrated in Fig. 1. By obtaining the discrete logarithm of any of the embedded elements through forgery and representations, we demonstrate that the DL problem solution can be found in either simulation $\mathcal{R}_1$ or $\mathcal{R}_2$.

Table 1. Simulations of the BLS-IBS scheme

Elements	Simulation $\mathcal{R}_1$	Simulation $\mathcal{R}_2$
X	g^a	g^x
H_{ID_i}	$g^{h_{ID_i}}$	$g^{au_{1,i}+v_{1,i}}$
H_{m_i}	$g^{h_{m_i}}$	$g^{au_{2,i}+v_{2,i}}$
d_{ID_i}	$g^{ah_{ID_i}}$	$g^{au_{1,i}x+v_{1,i}x}$
$\sigma^{(1)}_{ID_i,m_i}$	$g^{a(h_{ID_i}+h_{m_i}s_i)+h_{m_i}t_i}$	$g^{a(u_{1,i}x+u_{2,i}r_i)+v_{1,i}x+v_{2,i}r_i}$
$\sigma^{(2)}_{ID_i,m_i}$	$g^{as_i+t_i}$	g^{r_i}

In the security of the BLS-IBS scheme, for some randomly chosen x, h_{ID_i}, h_{m_i}, $r_i \in \mathbb{Z}_p^*$, let $X = g^x, H_{ID_i} = g^{h_{ID_i}}, H_{m_i} = g^{h_{m_i}}$, and $\sigma^{(2)}_{ID_i,m_i} = g^{r_i}$. It is easy to see every user private key d_{ID_i}, including that of the forged identity ID^*, is capable of being simulated in both simulations $\mathcal{R}_1$ and $\mathcal{R}_2$. As a result, our simulator will not abort during the query phase of the EUF-CMA security game as it is capable of responding to any user private key query or signature query.

During the forgery phase of the security game in the AGM and EUF-CMA security model, the algebraic adversary returns a forged signature $\sigma_{ID^*,m^*} = (\sigma^{(1)}_{ID^*,m^*}, \sigma^{(2)}_{ID^*,m^*}) = ((H_{ID^*})^x \cdot (H_{m^*})^{r^*}, g^{r^*})$ on a chosen identity and message pair (ID^*, m^*) and its representation vectors $[\vec{\alpha}], [\vec{\beta}]$ in $\mathbb{Z}_p$. Specifically, for some randomness $r^* \in \mathbb{Z}_p^*$, forgery σ_{ID^*,m^*} is defined as

$$\sigma^{(1)}_{ID^*,m^*} = g^{h_{ID^*}x+h_{m^*}r^*}, \quad \sigma^{(2)}_{ID^*,m^*} = g^{r^*}.$$

And forgery σ_{ID^*,m^*} can also be algebraically described with the corresponding representation vectors $[\vec{\alpha}], [\vec{\beta}]$ in $\mathbb{Z}_p$, such that

$$\begin{aligned}\sigma^{(1)}_{ID^*,m^*} =& g^{\alpha_0} X^{\alpha_1} \prod_i^{q_{h1}} (H_{ID_i})^{\alpha_{2,i}} \prod_i^{q_{h2}} (H_{m_i})^{\alpha_{3,i}} \prod_i^{q_e} (d_{ID_i})^{\alpha_{4,i}} \\ & \prod_i^{q_s} (\sigma^{(1)}_{ID^*,m_i})^{\alpha_{5,i}} (\sigma^{(2)}_{ID^*,m_i})^{\alpha_{6,i}}, \\ \sigma^{(2)}_{ID^*,m^*} =& g^{\beta_0} X^{\beta_1} \prod_i^{q_{h1}} (H_{ID_i})^{\beta_{2,i}} \prod_i^{q_{h2}} (H_{m_i})^{\beta_{3,i}} \prod_i^{q_e} (d_{ID_i})^{\beta_{4,i}} \\ & \prod_i^{q_s} (\sigma^{(1)}_{ID^*,m_i})^{\beta_{5,i}} (\sigma^{(2)}_{ID^*,m_i})^{\beta_{6,i}},\end{aligned}$$

where q_{h_1} is number of identity hash queries, q_{h_2} is number of message hash queries, q_e is number of extraction queries, and q_s is number of signing queries. Note that since the private keys of all users can be simulated, it is possible to exclude signatures queries for $(ID_i, \cdot)$ because the adversary can compute the signatures themselves with the knowledge of the user private keys d_{ID_i}, except for d_{ID^*} which the adversary is restricted to querying in the EUF-CMA security model.

Using the above definitions, the simulator can derive the following two modular equations.

$$\begin{aligned}h_{ID^*}x + h_{m^*}r^* =& \alpha_0 + x\alpha_1 + \sum_i^{q_{h1}} h_{ID_i}\alpha_{2,i} + \sum_i^{q_{h2}} h_{m_i}\alpha_{3,i} + \\ & \sum_i^{q_e} h_{ID_i}x\alpha_{4,i} + \sum_i^{q_s} (h_{ID^*}x + h_{m_i}r_i)\alpha_{5,i} + r_i\alpha_{6,i}, \\ r^* =& \beta_0 + x\beta_1 + \sum_i^{q_{h1}} h_{ID_i}\beta_{2,i} + \sum_i^{q_{h2}} h_{m_i}\beta_{3,i} + \\ & \sum_i^{q_e} h_{ID_i}x\beta_{4,i} + \sum_i^{q_s} (h_{ID^*}x + h_{m_i}r_i)\beta_{5,i} + r_i\beta_{6,i}.\end{aligned}$$

By rearranging with the term x and some $r_i \in [q_s]$, they can be simplified as

$$h_{ID^*}x + h_{m^*}r^* = x\theta + \hat{\theta} + \sum_i^{q_s} r_i\omega_{\alpha_i}, \tag{1}$$

$$r^* = x\delta + \hat{\delta} + \sum_i^{q_s} r_i\omega_{\beta_i}, \tag{2}$$

where $\delta, \hat{\delta}, \theta, \hat{\theta}, \omega_{\alpha_i}, \omega_{\beta_i} \in \mathbb{Z}_p$ are obtained from representation vectors $[\vec{\alpha}], [\vec{\beta}]$ in $\mathbb{Z}_p$ and elements $h_{ID_i}, h_{ID^*}, h_{m_i}, h_{ID^*}$ in $\mathbb{Z}_p^*$. We defer the definition in the full proof later. By using Eq. (1) and (2), we derive a new modular equation

$$x(h_{ID^*} + h_{m^*}\delta - \theta) + \sum_{i}^{q_s} r_i \Big(h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i} \Big) = \hat{\theta} - h_{m^*}\hat{\delta}.$$

Because the algebraic adversary is adaptive, it is necessary to analyze all the potential outcomes of their forgeries and representations. Figure 1 provides a diagrammatic overview of how different cases are classified, and how either (x, r_i) or $(h_{ID_i}, h_{m_i}, h_{ID^*}, h_{m^*})$ can be extracted in $\mathbb{Z}_p^*$ through simulation $\mathcal{R}_1$ or $\mathcal{R}_2$. Suppose adversary behaviours $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\mathbf{C}}, \overline{\mathbf{D}}$, as described in Fig. 1, are classified into condition $\mathcal{F}_1, \mathcal{F}_2$, such that $\mathcal{F}_1 : \mathbf{A} \vee \mathbf{B}$ and $\mathcal{F}_2 : (\overline{\mathbf{A}} \wedge \overline{\mathbf{B}}) \wedge (\mathbf{C} \vee \mathbf{D} \vee (\overline{\mathbf{C}} \wedge \overline{\mathbf{D}}))$. We see that all behaviours are captured, i.e. $\mathcal{F}_1 \vee \mathcal{F}_2 = 1$.

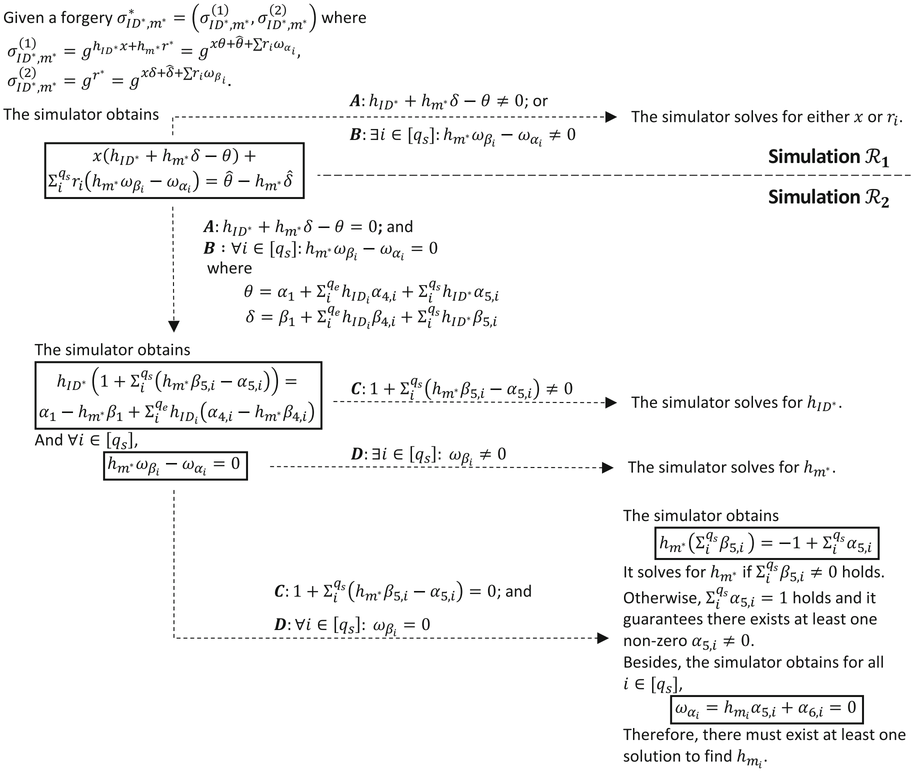

Given a forgery $\sigma^*_{ID^*,m^*} = \left(\sigma^{(1)}_{ID^*,m^*}, \sigma^{(2)}_{ID^*,m^*}\right)$ where
$\sigma^{(1)}_{ID^*,m^*} = g^{h_{ID^*}x + h_{m^*}r^*} = g^{x\theta + \hat{\theta} + \Sigma r_i \omega_{\alpha_i}}$,
$\sigma^{(2)}_{ID^*,m^*} = g^{r^*} = g^{x\delta + \hat{\delta} + \Sigma r_i \omega_{\beta_i}}$.
The simulator obtains
$\boxed{x(h_{ID^*} + h_{m^*}\delta - \theta) + \Sigma_i^{q_s} r_i(h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i}) = \hat{\theta} - h_{m^*}\hat{\delta}}$
$\boldsymbol{A}$: $h_{ID^*} + h_{m^*}\delta - \theta \neq 0$; or
$\boldsymbol{B}$: $\exists i \in [q_s]$: $h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i} \neq 0$
The simulator solves for either x or r_i.
Simulation $\mathcal{R}_1$
Simulation $\mathcal{R}_2$
$\overline{\boldsymbol{A}}$: $h_{ID^*} + h_{m^*}\delta - \theta = 0$; and
$\overline{\boldsymbol{B}}$: $\forall i \in [q_s]$: $h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i} = 0$
where
$\theta = \alpha_1 + \Sigma_i^{q_e} h_{ID_i}\alpha_{4,i} + \Sigma_i^{q_s} h_{ID^*}\alpha_{5,i}$
$\delta = \beta_1 + \Sigma_i^{q_e} h_{ID_i}\beta_{4,i} + \Sigma_i^{q_s} h_{ID^*}\beta_{5,i}$
The simulator obtains
$\boxed{h_{ID^*}\left(1 + \Sigma_i^{q_s}(h_{m^*}\beta_{5,i} - \alpha_{5,i})\right) = \alpha_1 - h_{m^*}\beta_1 + \Sigma_i^{q_e} h_{ID_i}(\alpha_{4,i} - h_{m^*}\beta_{4,i})}$
And $\forall i \in [q_s]$,
$\boxed{h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i} = 0}$
$\boldsymbol{C}$: $1 + \Sigma_i^{q_s}(h_{m^*}\beta_{5,i} - \alpha_{5,i}) \neq 0$
The simulator solves for h_{ID^*}.
$\boldsymbol{D}$: $\exists i \in [q_s]$: $\omega_{\beta_i} \neq 0$
The simulator solves for h_{m^*}.
$\overline{\boldsymbol{C}}$: $1 + \Sigma_i^{q_s}(h_{m^*}\beta_{5,i} - \alpha_{5,i}) = 0$; and
$\overline{\boldsymbol{D}}$: $\forall i \in [q_s]$: $\omega_{\beta_i} = 0$
The simulator obtains
$\boxed{h_{m^*}(\Sigma_i^{q_s}\beta_{5,i}) = -1 + \Sigma_i^{q_s}\alpha_{5,i}}$
It solves for h_{m^*} if $\Sigma_i^{q_s}\beta_{5,i} \neq 0$ holds.
Otherwise, $\Sigma_i^{q_s}\alpha_{5,i} = 1$ holds and it guarantees there exists at least one non-zero $\alpha_{5,i} \neq 0$.
Besides, the simulator obtains for all $i \in [q_s]$,
$\boxed{\omega_{\alpha_i} = h_{m_i}\alpha_{5,i} + \alpha_{6,i} = 0}$
Therefore, there must exist at least one solution to find h_{m_i}.

Fig. 1. A diagrammatic representation of the security proof in the AGM

Based on the definition of simulations $\mathcal{R}_1, \mathcal{R}_2$ as defined in Table 1, we see every element $x, r_i, h_{ID_i}, h_{m_i}$, h_{ID^*}, h_{m^*} has been implicitly embedded with unknown DL problem solution a, and known randomness $s_i, t_i, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, u_1^*, v_1^*, u_2^*, v_2^* \in \mathbb{Z}_p^*$. It is easy to see they are perfectly hidden from the view of the adversary, such that

$$\begin{aligned} x &= a, & r_i &= as_i + t_i, \\ h_{ID_i} &= au_{1,i} + v_{1,i}, & h_{m_i} &= au_{2,i} + v_{2,i}, \\ h_{ID^*} &= au_1^* + v_1^*, & h_{m^*} &= au_2^* + v_2^*. \end{aligned}$$

Therefore, the simulator manages to extract DL solution a as long as it obtains any of them $x, r_i, h_{ID_i}, h_{m_i}, h_{ID^*}, h_{m^*}$ in $\mathbb{Z}_p^*$. The full proof is detailed as follows.

Theorem 1. *Suppose hash functions $\mathcal{H}_1, \mathcal{H}_2$ are random oracles. If the DL problem is hard, the BLS-IBS scheme is secure against the EUF-CMA in the AGM with random oracles with a loss factor of* 2.

Proof. Suppose there exists an algebraic adversary who can (t, q_e, q_s, ϵ)-break the BLS-IBS scheme in the EUF-CMA model. We construct a simulator to solve the DL problem. Given as input a problem instance (g, g^a) over the pairing group tuple $\mathbb{PG} = (p, g, e, \mathbb{G}, \mathbb{G}_T)$, the simulator controls the random oracles $\mathcal{H}_1, \mathcal{H}_2$ and runs the algebraic adversary in a randomly chosen simulation $\mathcal{R}_1$ or $\mathcal{R}_2$ under condition $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively.

Simulation $\mathcal{R}_1$: Suppose the algebraic adversary returns forgery on (ID^*, m^*) under condition $\mathcal{F}_1$, given the DL problem instance g^a, the simulation is programmed by embedding g^a to master public key X and every queried signature element $\sigma^{(2)}_{ID^*, m_i} = g^{r_i}$ as follows.

Setup. The simulator sets $X = g^a$. The master public key is returned $mpk = (g, p, e, \mathbb{G}, \mathbb{G}_T, \mathcal{H}_1, \mathcal{H}_2, X)$.

Hash Query. At the beginning, the simulator prepares two empty sets L_{H1}, L_{H2} to record all hash queries and responses as follows.

$\mathcal{H}_1(ID_k)$: On an i-th random oracle query (ID_i). If $\langle ID_i, H_{ID_i}, h_{ID_i}\rangle \in L_{H1}$, the simulator responds to this query following the record. Otherwise, the simulator randomly selects $h_{ID_i} \in \mathbb{Z}_p$ and sets

$$H_{ID_i} = g^{h_{ID_i}}.$$

It programs $\mathcal{H}_1(ID_i) = H_{ID_i}$ and stores $\langle ID_k, H_{ID_i}, h_{ID_i}\rangle$ into L_{H1}. The simulator returns $\mathcal{H}_1(ID_i) = H_{ID_i}$.

$\mathcal{H}_2(m_i)$: On an i-th random oracle query (m_i). If $\langle m_i, H_{m_i}, h_{m_i}\rangle \in L_{H2}$, the simulator responds to this query following the record. Otherwise, the simulator randomly selects $h_{m_i} \in \mathbb{Z}_p$ and sets

$$H_{m_i} = g^{h_{m_i}}.$$

It programs $\mathcal{H}_2(m_i) = H_{m_i}$ and stores $\langle m_i, H_{m_i}, h_{m_i}\rangle$ into L_{H2}. The simulator returns $\mathcal{H}_2(m_i) = H_{m_i}$.

Query. The adversary makes extraction queries and signing queries in this phase. The simulator prepares an empty set L_E to record all queries and responses as follows.

$\mathcal{O}_E(ID_i)$: On an i-th extraction query (ID_i), the simulator checks whether $\langle ID_i, d_{ID_i}\rangle \in L_E$ exits. If there is, the simulator retrieves user private key d_{ID_i} which is based the definition as follows. Otherwise, it calls $\mathcal{H}_1(ID_i) \rightarrow H_{ID_i} = g^{h_{ID_i}}$, retrieves h_{ID_i} from L_{H1}, and sets

$$d_{ID_i} = (g^a)^{h_{ID_i}}.$$

The simulator stores $\langle ID_i, d_{ID_i}\rangle$ into L_E. The user private key d_{ID_i} is returned.

Correctness. It is easy to see d_{ID_i} is a valid user private key, such that

$$d_{ID_i} = (g^a)^{h_{ID_i}} = \mathcal{H}_1(ID_i)^x.$$

$\mathcal{O}_S(ID_i, m_i)$: On an i-th signing query (ID_i, m_i). It checks whether $\langle ID_i, d_{ID_i}\rangle \in L_E$. It calls $\mathcal{O}_E(ID_i) \rightarrow d_{ID_i}$ to generate fresh user private key if it does not exist. Otherwise, it retrieves d_{ID_i} from L_E. The simulator calls $\mathcal{H}_2(m_i) \rightarrow H_{m_i} = g^{h_{m_i}}$ and retrieves h_{m_i} from L_{H2}, randomly chooses $s_i, t_i \in \mathbb{Z}_p$, and sets signature $\sigma_{ID_i,m_i} = (\sigma^{(1)}_{ID_i,m_i}, \sigma^{(2)}_{ID_i,m_i})$ as follows. Let $r_i = as_i + t_i$,

$$\begin{aligned}\sigma^{(1)}_{ID_i,m_i} &= g^{a(h_{ID_i}+s_ih_{m_i})}g^{t_ih_{m_i}},\\ \sigma^{(2)}_{ID_i,m_i} &= g^{r_i} = g^{as_i+t_i}.\end{aligned}$$

Correctness. We see σ_{ID_i,m_i} is a valid signature on (ID_i, m_i), such that

$$\begin{aligned}\sigma^{(1)}_{ID_i,m_i} &= g^{a(h_{ID_i}+s_ih_{m_i})}g^{t_ih_{m_i}}\\ &= g^{ah_{ID_i}}g^{h_{m_i}(as_i+t_i)}\\ &= d_{ID_i}\cdot \mathcal{H}_2(m_i)^{r_i}.\end{aligned}$$

Forgery. Assume the forgery is returned under condition $\mathcal{F}_1$. The algebraic adversary returns a forged signature $\sigma_{ID^*,m^*} = (\sigma^{(1)}_{ID^*,m^*}, \sigma^{(2)}_{ID^*,m^*})$ on a chosen identity and message pair (ID^*, m^*) following the definition, such that for some $r^* \in \mathbb{Z}_p$,

$$\begin{aligned}\sigma^{(1)}_{ID^*,m^*} &= (H_1^*)^x(H_2^*)^{r^*}\\ &= g^{h_{ID^*}x+h_{m^*}r^*},\\ \sigma^{(2)}_{ID^*,m^*} &= g^{r^*}.\end{aligned}$$

It also attaches together with some representation vectors $[\vec{\alpha}], [\vec{\beta}]$ in $\mathbb{Z}_p$, such that

$$\begin{aligned}[\vec{\alpha}] = (&\alpha_0, \alpha_1, \alpha_{2,1}, \cdots, \alpha_{2,q_{h1}}, \alpha_{3,1}, \cdots, \alpha_{3,q_{h2}}, \alpha_{4,1}, \\ &\cdots, \alpha_{4,q_e}, q_{5,1}, \cdots, \alpha_{5,q_s}, \alpha_{6,1}, \cdots, \alpha_{6,q_s}), \\ [\vec{\beta}] = (&\beta_0, \beta_1, \beta_{2,1}, \cdots, \beta_{2,q_{h1}}, \beta_{3,1}, \cdots, \beta_{3,q_{h2}}, \beta_{4,1}, \\ &\cdots, \beta_{4,q_e}, q_{5,1}, \cdots, \beta_{5,q_s}, \beta_{6,1}, \cdots, \beta_{6,q_s}),\end{aligned}$$

which indicates how the forgery being algebraically computed, i.e.,

$$\begin{aligned}\sigma^{(1)}_{ID^*,m^*} =& g^{\alpha_0} X^{\alpha_1} \prod_i^{q_{h1}} (H_{ID_i})^{\alpha_{2,i}} \prod_i^{q_{h2}} (H_{m_i})^{\alpha_{3,i}} \\ & \prod_i^{q_e} (d_{ID_i})^{\alpha_{4,i}} \prod_i^{q_s} (\sigma^{(1)}_{ID^*,m_i})^{\alpha_{5,i}} (\sigma^{(2)}_{ID^*,m_i})^{\alpha_{6,i}}, \\ \sigma^{(2)}_{ID^*,m^*} =& g^{\beta_0} X^{\beta_1} \prod_i^{q_{h1}} (H_{ID_i})^{\beta_{2,i}} \prod_i^{q_{h2}} (H_{m_i})^{\beta_{3,i}} \\ & \prod_i^{q_e} (d_{ID_i})^{\beta_{4,i}} \prod_i^{q_s} (\sigma^{(1)}_{ID^*,m_i})^{\beta_{5,i}} (\sigma^{(2)}_{ID^*,m_i})^{\beta_{6,i}}.\end{aligned}$$

Note that we omit signature query for any $\mathcal{O}_S(ID_i, \cdot)$ as the adversary may obtain $d_{ID_i} \leftarrow \mathcal{O}_E(ID_i)$ from the extraction oracle and sign the signature by itself. In order to simplify the above definition, let $\omega_{\alpha_i} = h_{m_i}\alpha_{5,i} + \alpha_{6,i}$, and $\omega_{\beta_i} = h_{m_i}\beta_{5,i} + \beta_{6,i}$,

$$\begin{aligned}\theta = \alpha_1 + \sum_i^{q_e} h_{ID_i}\alpha_{4,i} + \sum_i^{q_s} h_{ID^*}\alpha_{5,i}, \quad \hat{\theta} = \alpha_0 + \sum_i^{q_{h1}} h_{ID_i}\alpha_{2,i} + \sum_i^{q_{h2}} h_{m_i}\alpha_{3,i}, \\ \delta = \beta_1 + \sum_i^{q_e} h_{ID_i}\beta_{4,i} + \sum_i^{q_s} h_{ID^*}\beta_{5,i}, \quad \hat{\delta} = \beta_0 + \sum_i^{q_{h1}} h_{ID_i}\beta_{2,i} + \sum_i^{q_{h2}} h_{m_i}\beta_{3,i}.\end{aligned}$$

Based on above definition, it is easy to see $\sigma_{ID^*,m^*} = (\sigma^{(1)}_{ID^*,m^*}, \sigma^{(2)}_{ID^*,m^*})$ can be simply written as follows, such that

$$\sigma^{(1)}_{ID^*,m^*} = g^{x\theta + \hat{\theta} + \sum_i^{q_s} r_i(\omega_{\alpha_i})}, \quad \sigma^{(2)}_{ID^*,m^*} = g^{x\delta + \hat{\delta} + \sum_i^{q_s} r_i(\omega_{\beta_i})},$$

which yields the following two modular equations

$$\begin{aligned}h_{ID^*}x + h_{m^*}r^* &= x\theta + \hat{\theta} + \sum_i^{q_s} r_i\omega_{\alpha_i}, \\ r^* &= x\delta + \hat{\delta} + \sum_i^{q_s} r_i\omega_{\beta_i}.\end{aligned}$$

Therefore, given the forgery and representation vectors, the simulator obtains the following modular equation by substituting r^*, such that

$$x(h_{ID^*} + h_{m^*}\delta - \theta) + \sum_{i}^{q_s} r_i(h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i}) = \hat{\theta} - h_{m^*}\hat{\delta}$$

Abort. The simulator aborts if $\overline{\mathbf{A}} : h_{ID^*} + h_{m^*}\delta - \theta = 0$ and $\overline{\mathbf{B}} : \forall i \in [q_s] : h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i} = 0$ hold.

Otherwise, based on above simulation definitions, where $x = a$ and $\forall i \in [q_s] : r_i = as_i + t_i$, the simulator can solve for a under condition $\mathcal{F}_1 : \mathbf{A} \vee \mathbf{B}$, such that behaviour $\mathbf{A} : h_{ID^*} + h_{m^*}\delta - \theta \neq 0$ or $\mathbf{B} : \exists i \in [q_s] : h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i} \neq 0$ holds. We can find a by computing

$$a = \frac{\hat{\theta} - h_{m^*}\hat{\delta} + \sum_i^{q_s} t_i(\omega_{\alpha_i} - h_{m^*}\omega_{\beta_i})}{h_{ID^*} + h_{m^*}\delta - \theta + \sum_i^{q_s} s_i(h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i})}.$$

Success Probability of $\mathcal{R}_1$. There is no abort in the simulation as the simulator manages to respond to every extraction and signing query. By condition $\mathcal{F}_1$, the simulator must obtain reducible forgery and representation as either $\mathbf{A} : h_{ID^*} + h_{m^*}\delta - \theta \neq 0$ or $\mathbf{B} : \sum_i^{q_s} s_i(h_{m^*}\omega_{\beta_i} - \omega_{\alpha_i}) \neq 0$ must hold. In particular, the success probability $\Pr[\text{Success}|\mathcal{R}_1]$ to extract the DL solution in $\mathcal{R}_1$ is based condition $\mathcal{F}_1$, hence $\Pr[\text{Success}|\mathcal{R}_1]$ is described as follows.

$$\begin{aligned}\Pr[\text{Success}|\mathcal{R}_1] &= \Pr[\text{Success}|\mathcal{F}_1]\\ &= \Pr[\mathbf{A} \vee \mathbf{B}]\\ &= 1 - (\Pr[\overline{\mathbf{A}}] \wedge \Pr[\overline{\mathbf{B}}])\\ &= \frac{3}{4}\end{aligned}$$

Simulation $\mathcal{R}_2$: Suppose the algebraic adversary returns forgery on (ID^*, m^*) under condition $\mathcal{F}_2$, the simulation is programmed by embedding the DL problem instance g^a to each hash query H_{ID_i}, H_{m_i}, including H_{ID^*}, H_{m^*}.

Setup. The simulator randomly chooses $x \in \mathbb{Z}_p$ and sets $X = g^x$. The master public key is returned $mpk = (g, p, e, \mathbb{G}, \mathbb{G}_T, \mathcal{H}_1, \mathcal{H}_2, X)$.

Hash Query. The simulator prepares two empty sets L_{H1}, L_{H2} to record all queries and responses.

$\mathcal{H}_1(ID_i)$: On an i-th random oracle query (ID_i). If $\langle ID_i, H_{ID_i}, u_{1,i}, v_{1,i}\rangle \in L_{H1}$, the simulator responds to this query following the record. Otherwise, the simulator randomly selects $u_{1,i}, v_{1,i} \in \mathbb{Z}_p$ and sets

$$H_{ID_i} = g^{au_{1,i}+v_{1,i}}.$$

It programs $\mathcal{H}_1(ID_i) = H_{ID_i}$ and stores $\langle ID_i, H_{ID_i}, u_{1,i}, v_{1,i}\rangle$ into L_{H1}. The simulator returns $\mathcal{H}_1(ID_i) = H_{ID_i}$.

$\mathcal{H}_2(m_i)$: On an i-th random oracle query (m_i). If $\langle m_i, H_{m_i}, u_{2,i}, v_{2,i}\rangle \in L_{H2}$, the simulator responds to this query following the record. Otherwise, the simulator randomly selects $u_{2,i}, v_{2,i} \in \mathbb{Z}_p$ and sets

$$H_{m_i} = g^{au_{2,i}+v_{2,i}}.$$

It programs $\mathcal{H}_2(m_i) = H_{m_i}$ and stores $\langle m_i, H_{m_i}, u_{2,i}, v_{2,i}\rangle$ into L_{H2}. The simulator returns $\mathcal{H}_2(m_i) = H_{m_i}$.

Query. Similar to the definition of $\mathcal{R}_1$, the adversary makes extraction queries and signing queries in this phase. The simulator prepares an empty set L_E to record all queries and responses as follows.

$\mathcal{O}_E(ID_i)$: On an i-th extraction query (ID_i), the simulator checks whether $\langle ID_i, d_{ID_i}\rangle \in L_E$ exists. If there is, the simulator retrieves user private key d_{ID_i} which is based the definition as follows. Otherwise, it calls $\mathcal{H}_1(ID_i) \rightarrow H_{ID_i} = g^{au_{1,i}+v_{1,i}}$ and retrieves $u_{1,i}, v_{1,i} \in \mathbb{Z}_p$ from L_{H1}, and sets $d_{ID_i} = g^{au_{1,i}x+v_{1,i}x}$. The simulator stores $\langle ID_i, d_{ID_i}\rangle$ into L_E. The user private key d_{ID_i} is returned.

Correctness. It is easy to see d_{ID_i} is a valid user private key, such that

$$\begin{aligned} d_{ID_i} &= g^{au_{1,i}x+v_{1,i}x} \\ &= \mathcal{H}_1(ID_i)^x \end{aligned}$$

$\mathcal{O}_S(ID_i, m_i)$: On an i-th signing query (ID_i, m_i). It checks whether $\langle ID_i, d_{ID_i}\rangle \in L_E$. It calls $\mathcal{O}_E(ID_i) \rightarrow d_{ID_i}$ to generate fresh user private key if it does not exist. Otherwise, it retrieves d_{ID_i}. The simulator calls $\mathcal{H}_2(m_i) = H_{m_i} = g^{au_{2,i}+v_{2,i}}$ and retrieves $u_{2,i}, v_{2,i}$ from L_{H2}, randomly chooses $r_i \in \mathbb{Z}_p$, and sets signature $\sigma_{ID_i,m_i} = (\sigma^{(1)}_{ID_i,m_i}, \sigma^{(2)}_{ID_i,m_i})$ as follows

$$\begin{aligned} \sigma^{(1)}_{ID_i,m_i} &= g^{a(xu_{1,i}+r_iu_{2,i})}g^{xv_1 1,i+r_iv_{2,i}}, \\ \sigma^{(2)}_{ID_i,m_i} &= g^{r_i}. \end{aligned}$$

Correctness. It is easy to verify σ_{ID_i,m_i} is a valid signature, such that

$$\begin{aligned} \sigma^{(1)}_{ID_i,m_i} &= g^{a(xu_{1,i}+r_iu_{2,i})}g^{xv_{1,i}+r_iv_{2,i}} \\ &= g^{x(au_{1,i}+v_{1,i})}g^{r_i(au_{2,i}+v_{2,i})} \\ &= d_{ID_i} \cdot \mathcal{H}_2(m_i)^{r_i} \end{aligned}$$

Forgery. Assume the forgery is returned under condition $\mathcal{F}_2$. The algebraic adversary returns a forged signature $\sigma_{ID^*,m^*} = (\sigma^{(1)}_{ID^*,m^*}, \sigma^{(2)}_{ID^*,m^*})$ on a chosen identity and message pair (ID^*, m^*) along with the representation vectors $[\vec{\alpha}], [\vec{\beta}]$ in $\mathbb{Z}_p$ following the definition as in $\mathcal{R}_1$. Let

$$g^{\theta} = g^{\alpha_1}\prod_{i}^{q_e}(H_{ID_i})^{\alpha_{4,i}}\prod_{i}^{q_s}(H_{ID^*})^{\alpha_{5,i}}, \quad g^{\delta} = g^{\beta_1}\prod_{i}^{q_e}(H_{ID_i})^{\beta_{4,i}}\prod_{i}^{q_s}(H_{ID^*})^{\beta_{5,i}},$$

$$g^{\omega_{\alpha_i}} = (H_{m_i})^{\alpha_{5,i}}g^{\alpha_{6,i}}, \qquad g^{\omega_{\beta_i}} = (H_{m_i})^{\beta_{5,i}}g^{\beta_{6,i}}.$$

Abort. The simulator aborts if $e(H_{ID^*}, g)\ e(H_{m^*}, g^{\delta}) \neq e(g, g^{\theta})$ holds and there exist some $i \in [q_s]$, such that $e(H_{m^*}, g^{\omega_{\beta_i}}) \neq e(g, g^{\omega_{\alpha_i}})$.

Based on above simulation definitions, where $\forall i \in [q_{h1}] : h_{ID_i} = au_{1,i} + v_{1,i}$ and $\forall i \in [q_{h2}] : h_{m_i} = au_{2,i} + v_{2,i}$, the simulator can solve for a under condition $\mathcal{F}_2 : (\overline{\mathbf{A}} \wedge \overline{\mathbf{B}}) \wedge (\mathbf{C} \vee \mathbf{D} \vee (\overline{\mathbf{C}} \wedge \overline{\mathbf{D}}))$ in the following three cases. It is worth noting that due to behaviour $\overline{\mathbf{A}} \wedge \overline{\mathbf{B}}$, the simulator must obtains following two equations

$$\overline{\mathbf{A}} : h_{ID^*} + h_{m^*}\delta = \theta \tag{3}$$

$$\overline{\mathbf{B}} : \forall i \in [q_s] : h_{m^*}\omega_{\beta_i} = \omega_{\alpha_i} \tag{4}$$

The simulator runs **Case 1** if $1 \neq g \cdot \prod_i^{q_s}((H_{m^*})^{\beta_{5,i}} \cdot g^{-\alpha_{5,i}})$ holds.
Case 1: $(\overline{\mathbf{A}} \wedge \overline{\mathbf{B}}) \wedge \mathbf{C})$. The simulator obtains Eq. (3): $h_{ID^*} + h_{m^*}\delta = \theta$ where $\theta = \alpha_1 + \sum_i^{q_e} h_{ID_i}\alpha_{4,i} + \sum_i^{q_s} h_{ID^*}\alpha_{5,i}$ and $\delta = \beta_1 + \sum_i^{q_e} h_{ID_i}\beta_{4,i} + \sum_i^{q_s} h_{ID^*}\beta_{5,i}$, which yields

$$\begin{aligned} &h_{ID^*}\,(1 + \varSigma_i^{q_s}(h_{m^*}\beta_{5,i} - \alpha_{5,i})) = \\ &\alpha_1 - h_{m^*}\beta_1 + \varSigma_i^{q_e} h_{ID_i}(\alpha_{4,i} - h_{m^*}\beta_{4,i}). \end{aligned}$$

Based on the definition of simulation $\mathcal{R}_2$, the simulator checks if $\exists i \in [q_e]$: $\beta_{4,i} \neq 0$ or $\varSigma_i^{q_s}\beta_{5,i} \neq 0$ hold. The simulator manages to derive the following modular quadratic equation

$$a^2\varDelta + a\varDelta' + \varDelta'' = 0,$$

such that $\varDelta, \varDelta', \varDelta''$ in $\mathbb{Z}_p$ can be defined as follows, which are computed based on all chosen randomness $u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, u_1^*, v_1^*, u_2^*, v_2^* \in \mathbb{Z}_p^*$ by the simulator and received representation vectors $[\vec{\alpha}], [\vec{\beta}]$ in $\mathbb{Z}_p$,

$$\begin{aligned} \varDelta &= \varSigma_i^{q_e} u_2^* u_{1,i}\beta_{4,i} + \varSigma_i^{q_s} u_2^* u_1^* \beta_{5,i}, \\ \varDelta' &= u_1^* + u_2^*\beta_1 + \varSigma_i^{q_e}(u_2^* v_{1,i}\beta_{4,i} + v_2^* u_{1,i}\beta_{4,i} - u_{1,i}\alpha_{4,i}) \\ &\quad + \varSigma_i^{q_s}(u_2^* v_1^* \beta_{5,i} + v_2^* u_1^* \beta_{5,i} - u_1^*\alpha_{5,i}), \\ \varDelta'' &= v_1^* + v_2^*\beta_1 - \alpha_1 + \varSigma_i^{q_e} v_{1,i}(v_2^*\beta_{4,i} - \alpha_{4,i}) \\ &\quad + \varSigma_i^{q_s} v_1^*(v_2^*\beta_{5,i} - \alpha_{5,i}). \end{aligned}$$

The simulator can find at most two solutions a_0 or a_1 by solving the above modular quadratic equation. It can test for the correct solution via the given DL problem instance $g^a = g^{a_0}$ or $g^a = g^{a_1}$.
Suppose the simulator obtains $\forall i \in [q_e] : \beta_{4,i} = 0$ and $\varSigma_i^{q_s}\beta_{5,i} = 0$ which implicitly sets $\varDelta = 0$. In this case, the simulator solves for a by computing

$$a = -\frac{v_1^* + v_2^*\beta_1 - \alpha_1 + \varSigma_i^{q_e} v_{1,i}(-\alpha_{4,i}) + \varSigma_i^{q_s} v_1^*(-\alpha_{5,i})}{u_1^* + u_2^*\beta_1 + \varSigma_i^{q_e}(-u_{1,i}\alpha_{4,i}) + \varSigma_i^{q_s}(-u_1^*\alpha_{5,i})}.$$

Due to behaviour $\mathbf{C} : 1 + \Sigma_i^{q_s}(h_{m^*}\beta_{5,i} - \alpha_{5,i}) \neq 0$ holds, even though $\Sigma_i^{q_s}\beta_{5,i} = 0$ is set, it ensures that $\Sigma_i^{q_s}\alpha_{5,i} \neq 1$. While value u_1^* remains perfectly hidden due to random oracles, such that $h_{ID^*} = au_1^* + v_1^*$ is implicitly set, the adversary has no advantage to reproduce u_1^* if $\Sigma_i^{q_s}\alpha_{5,i} \neq 1$ holds. Therefore, the simulator must be able to find a as the success probability of setting $u_1^* + u_2^*\beta_1 + \Sigma_i^{q_e}(-u_{1,i}\alpha_{4,i}) + \Sigma_i^{q_s}(-u_1^*\alpha_{5,i}) = 0$ is negligible.
Otherwise, the simulator runs **Case 2** if there exists at least one i in $[q_s]$ that $H_{m_i}^{\beta_{5,i}} \cdot g^{\beta_{6,i}} \neq 1$.
Case 2: $(\overline{\mathbf{A}} \wedge \overline{\mathbf{B}}) \wedge \mathbf{D})$. The simulator retrieves Eq. (4):$\forall i \in [q_s] : h_{m^*}\omega_{\beta_i} = \omega_{\alpha_i}$, where $\omega_{\alpha_i} = h_{m_i}\alpha_{5,i} + \alpha_{6,i}$ and $\omega_{\beta_i} = h_{m_i}\beta_{5,i} + \beta_{6,i}$. Due to the definition of simulation $\mathcal{R}_2$, if $\exists i \in [q_s]\beta_{5,i} \neq 0$ holds, the simulator must obtain at least one modular quadratic equation

$$a^2\lambda + a\lambda' + \lambda'' = 0,$$

where $\lambda, \lambda', \lambda''$ in $\mathbb{Z}_p$ are computable by the simulator, such as

$$\begin{aligned}\lambda &= u_2^* u_{2,i}\beta_{5,i}\\ \lambda' &= u_2^*(v_{2,i}\beta_{5,i} + \beta_{6,i}) + u_{2,i}(v_2^*\beta_{5,i} - \alpha_{5,i})\\ \lambda'' &= v_2^*(v_{2,i}\beta_{5,i} + \beta_{6,i}) - v_{2,i}\alpha_{5,i} - \alpha_{6,i}.\end{aligned}$$

The simulator can find at most two solutions a_0 or a_1 by solving the above modular quadratic equation. It can test for the correct solution via the given DL problem instance $g^a = g^{a_0}$ or $g^a = g^{a_1}$. Suppose the simulator obtains $\forall i \in [q_s] : \beta_{5,i} = 0$, which implicitly sets $\lambda = 0$. The simulator can solve for a by computing

$$a = -\frac{v_2^*(\beta_{6,i}) - v_{2,i}\alpha_{5,i} - \alpha_{6,i}}{u_2^*(\beta_{6,i}) + u_{2,i}(-\alpha_{5,i})}.$$

Due to behaviour $\mathbf{D} : \exists i \in [q_s] : \omega_{\beta_i} = h_{m_i}\beta_{5,i} + \beta_{6,i} \neq 0$ holds, even though $\forall i \in [q_s] : \beta_{5,i} = 0$ is set, it ensures that there exists at least one $\beta_{6,i} \neq 0$. Again, value u_2^* is perfectly hidden due to the random oracles, such that $h_{m^*} = au_2^* + v_2^*$ is implicitly set, the adversary has no advantage to reproduce u_2^*. Therefore, the simulator must be able to find a as the success probability to set $\forall i \in [q_s] : v_2^*(\beta_{6,i}) - v_{2,i}\alpha_{5,i} - \alpha_{6,i} = 0$ is negligible.
Otherwise, behaviours $\overline{\mathbf{C}} \wedge \overline{\mathbf{D}}$ must hold, and they yield following new equations

$$\overline{\mathbf{C}} : h_{m^*}\sum_i^{q_s}\beta_{5,i} = -1 + \sum_i^{q_s}\alpha_{5,i} \tag{5}$$

$$\overline{\mathbf{D}} : \forall i \in [q_s] : \omega_{\alpha_i} = h_{m_i}\alpha_{5,i} + \alpha_{6,i} = 0 \tag{6}$$

The simulator runs **Case 3**.

Case 3: $(\overline{\mathbf{A}} \wedge \overline{\mathbf{B}}) \wedge (\overline{\mathbf{C}} \wedge \overline{\mathbf{D}})$. The simulator checks if $\sum_i^{q_s} \beta_{5,i} \neq 0$ holds. Due to the definition of simulation $\mathcal{R}_2$, the simulator retrieves Eq. (5):$h_{m^*} \sum_i^{q_s} \beta_{5,i} = -1 + \sum_i^{q_s} \alpha_{5,i}$ and solves for a by computing

$$a = \frac{-1 + \sum_i^{q_s} (\alpha_{5,i} - v_2^* \beta_{5,i})}{u_2^* \sum_i^{q_s} \beta_{5,i}}.$$

Otherwise, $\sum_i^{q_s} \beta_{5,i} \neq 0$ yields that $\sum_i^{q_s} \alpha_{5,i} = 1$, which implies there exists at least one non-zero $\alpha_{5,i} \neq 0$ in $[q_s]$. Recall that the simulator still obtains Eq. (6):$\forall i \in [q_s] : \omega_{\alpha_i} = h_{m_i} \alpha_{5,i} + \alpha_{6,i} = 0$. Therefore, there is at least one solution to find a by computing

$$a = -\frac{\alpha_{6,i} + v_{2,i} \alpha_{5,i}}{u_{2,i} \alpha_{5,i}}.$$

Success Probability of $\mathcal{R}_2$. There is no abort in the simulation as the simulator manages to respond to every extraction and signing query. By condition $\mathcal{F}_2$ where the simulator obtains Eq. (3) and (4) due to behaviour $\overline{\mathbf{A}} \wedge \overline{\mathbf{B}}$, $1 + \sum_i^{q_s} (h_{m^*} \beta_{5,i} - \alpha_{5,i}) \neq 0$ due to behaviour $\mathbf{C}$, $\exists i \in [q_s] : \sum_i^{q_s} \beta_{5,i} \neq 0$ due to behaviour $\mathbf{D}$, and Eq. (5) and (6) due to behaviour $\overline{\mathbf{C}} \wedge \overline{\mathbf{D}}$, the simulator must obtain reducible forgery and representation as defined in either cases, which ensures solving for a with an overwhelming success probability. In particular, the success probability $\Pr[\text{Success}|\mathcal{R}_2]$ to extract the DL solution in $\mathcal{R}_2$ is based condition $\mathcal{F}_2$, hence $\Pr[\text{Success}|\mathcal{R}_2]$ is described as follows.

$$\begin{aligned} \Pr[\text{Success}|\mathcal{R}_2] &= \Pr[\text{Success}|\mathcal{F}_2] \\ &= \Pr[\textbf{Case 1}] + \Pr[\textbf{Case 2}] + \Pr[\textbf{Case 3}] \\ &= \frac{1}{4} \end{aligned}$$

Indistinguishable Simulations. According to both simulations $\mathcal{R}_1$, $\mathcal{R}_2$, the simulations are correct due to the correctness of every simulated user private key and signature holds. In addition, the simulator sets all elements with perfectly hidden randomness. For example, master public key $X = g^x$, hash responses $H_{ID_i} = g^{h_{ID_i}}, H_{m_i} = g^{h_{m_i}}$, signature randomnesses $\sigma^{(2)}_{ID_i,m_i} = g^{r_i}$. They are, respectively, simulated as

$$\begin{cases} a, & h_{ID_i}, & h_{m_i}, & as_i + t_i : & \mathcal{R}_1 \\ x, & au_{1,i} + v_{1,i}, & au_{2,i} + v_{2,i}, & r_i \qquad : & \mathcal{R}_2 \end{cases}$$

where $a, h_{ID_i}, h_{m_i}, s_i, t_i, x, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, r_i \in \mathbb{Z}_p^*$ are all randomly chosen; therefore, all elements look random and independent from the point of view of the adversary.

Reduction Loss. The concrete security is calculated based on the success probability of the two simulations. Since the two simulations are indistinguishable, the simulator randomly selects $\mu \in \{0,1\}$ and runs either one of the simulations. We describe the success probability of extracting the DL solution as follows.

$$\begin{aligned}\Pr[\text{Success}] &= \Pr[\text{Success}|\mathcal{R}_1]\Pr[\mu=0] + \Pr[\text{Success}|\mathcal{R}_2]\Pr[\mu=1]\\ &= \frac{3}{4}(\frac{1}{2}) + \frac{1}{4}(\frac{1}{2}) = \frac{1}{2}\end{aligned}$$

The given forgery and representations can be reducible in either one of the simulations under the condition $\mathcal{F}_1$ or $\mathcal{F}_2$ with overwhelming success probability. Therefore, the success probability to extract DL problem solution is at least $\frac{1}{2}$, which concludes that the security loss of the BLS-IBS scheme under EUF-CMA and DL assumption is 2.

This completes the proof of Theorem 1. □

5 Conclusion

In this work, we achieved a breakthrough by demonstrating tight security for IBS in the AGM. In particular, we established the security of the BLS-IBS scheme, which is derived from BLS signatures, under the DL assumption and EUF-CMA with random oracles in the AGM. Our security proof involves two simulations that can simulate any user private key and signature. Although the forgery and representations by the algebraic adversary were classified into several cases, we were able to capture them with either of the two simulations, resulting in a reduction loss of two. The research outcome of this paper provides valuable insight into achieving tight security for IBS in the AGM.

Further research could examine the possibility of a pairing-free IBS scheme that provides ideal security. The Schnorr-like IBS by Galindo and Garcia is currently the most efficient IBS without pairing, however, we identified that its security cannot be proven tight even in the AGM as it is unable to both respond to private key queries and extract the problem solution from any forgery in a simulation.

Acknowledgement. We would like to thank the anonymous reviewers for their constructive comments. Fuchun Guo was supported by ARC Future Fellowship (FT220100046).

References

1. Abe, M., Groth, J., Ohkubo, M.: Separating short structure-preserving signatures from non-interactive assumptions. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 628–646. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_34
2. Bacho, R., Loss, J.: On the adaptive security of the threshold BLS signature scheme. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 193–207 (2022)

3. Barreto, P.S.L.M., Libert, B., McCullagh, N., Quisquater, J.-J.: Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 515–532. Springer, Heidelberg (2005). https://doi.org/10.1007/11593447_28
4. Bellare, M., Dai, W.: Chain reductions for multi-signatures and the HBMS scheme. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13093, pp. 650–678. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92068-5_22
5. Bellare, M., Namprempre, C., Neven, G.: Security proofs for identity-based identification and signature schemes. J. Cryptol. **22**(1), 1–61 (2009)
6. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, pp. 62–73 (1993)
7. Beth, T.: Efficient zero-knowledge identification scheme for smart cards. In: Barstow, D., et al. (eds.) EUROCRYPT 1988. LNCS, vol. 330, pp. 77–84. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-45961-8_7
8. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. J. Cryptol. **17**(4), 297–319 (2004)
9. Boneh, D., Venkatesan, R.: Breaking RSA may not be equivalent to factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 59–71. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054117
10. Bresson, E., Monnerat, J., Vergnaud, D.: Separation results on the "One-More" computational problems. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 71–87. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79263-5_5
11. Chatterjee, S., Kamath, C., Kumar, V.: Galindo-Garcia identity-based signature revisited. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 456–471. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37682-5_32
12. Chin, J.J., Tan, S.Y., Heng, S.H., Phan, R.C.W.: Twin-schnorr: a security upgrade for the schnorr identity-based identification scheme. Sci. World J. 2015 (2015)
13. Choon, J.C., Hee Cheon, J.: An identity-based signature from gap Diffie-Hellman groups. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 18–30. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36288-6_2
14. Coron, J.-S.: Optimal security proofs for PSS and other signature schemes. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 272–287. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_18
15. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
16. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_2
17. Fuchsbauer, G., Plouviez, A., Seurin, Y.: Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 63–95. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_3
18. Fukumitsu, M., Hasegawa, S.: A Galindo-Garcia-like identity-based signature with tight security reduction. In: 2017 Fifth International Symposium on Computing and Networking (CANDAR), pp. 87–93. IEEE (2017)
19. Fukumitsu, M., Hasegawa, S.: A Galindo-Garcia-like identity-based signature with tight security reduction, revisited. In: 2018 Sixth International Symposium on Computing and Networking (CANDAR), pp. 92–98. IEEE (2018)

20. Galindo, D., Garcia, F.D.: A Schnorr-like lightweight identity-based signature scheme. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 135–148. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02384-2_9
21. Garg, S., Bhaskar, R., Lokam, S.V.: Improved bounds on security reductions for discrete log based signatures. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 93–107. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_6
22. Goh, E.J., Jarecki, S., Katz, J., Wang, N.: Efficient signature schemes with tight reductions to the Diffie-Hellman problems. J. Cryptol. **20**(4), 493–514 (2007)
23. Hess, F.: Efficient identity based signature schemes based on pairings. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 310–324. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36492-7_20
24. Kastner, J., Loss, J., Xu, J.: On pairing-free blind signature schemes in the algebraic group model. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022. LNCS, vol. 13178, pp. 468–497. Springer, Cham (2022)
25. Kılınç Alper, H., Burdges, J.: Two-round trip Schnorr multi-signatures via delinearized witnesses. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 157–188. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_7
26. Kiltz, E., Masny, D., Pan, J.: Optimal security proofs for signatures from identification schemes. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 33–61. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_2
27. Kurosawa, K., Heng, S.-H.: From digital signature to ID-based identification/signature. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 248–261. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24632-9_18
28. Maurer, U.M., Wolf, S.: The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. SIAM J. Comput. **28**(5), 1689–1721 (1999)
29. Narayan, S., Parampalli, U.: Efficient identity-based signatures in the standard model. IET Inf. Secur. **2**(4), 108–118 (2008)
30. Nick, J., Ruffing, T., Seurin, Y.: MuSig2: simple two-round schnorr multi-signatures. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 189–221. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_8
31. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 31–53. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_3
32. Paillier, P., Vergnaud, D.: Discrete-log-based signatures may not be equivalent to discrete log. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 1–20. Springer, Heidelberg (2005). https://doi.org/10.1007/11593447_1
33. Paterson, K.G.: Id-based signatures from pairings on elliptic curves. Electron. Lett. **38**(18), 1025–1026 (2002)
34. Paterson, K.G., Schuldt, J.C.N.: Efficient identity-based signatures secure in the standard model. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 207–222. Springer, Heidelberg (2006). https://doi.org/10.1007/11780656_18
35. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptol. **13**(3), 361–396 (2000)

36. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
37. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_5
38. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_7
39. Yi, P.: An efficient identity-based signature scheme with provable security. Inf. Sci. **576**, 790–799 (2021)
40. Zhang, X., Liu, S., Gu, D., Liu, J.K.: A generic construction of tightly secure signatures in the multi-user setting. Theoret. Comput. Sci. **775**, 32–52 (2019)

Compact Password Authenticated Key Exchange from Group Actions

Ren Ishibashi and Kazuki Yoneyama(✉)

Ibaraki University, 4-12-1, Nakanarusawa, Hitachi, Ibaraki 316-8511, Japan
kazuki.yoneyama.sec@vc.ibaraki.ac.jp

Abstract. At ASIACRYPT 2020, Alamati et al. formalized the framework of group actions for abstracting isogeny-based cryptosystems. At CRYPTO 2022, Abdalla et al. extended the framework to represent the quadratic twist of elliptic curves, and proposed the first provably secure and tightly secure one-round isogeny-based password-authenticated key exchange (PAKE) scheme ($\mathsf{X\text{-}GA\text{-}PAKE}$) by a bit-by-bit approach. However, in $\mathsf{X\text{-}GA\text{-}PAKE}$, for the password length ℓ, the number of group actions per party is 5ℓ, and the communication complexity per party is 2ℓ, thus there is a problem in efficiency. In this paper, we propose an efficient one-round PAKE scheme that reduces the number of group actions and the communication complexity compared to $\mathsf{X\text{-}GA\text{-}PAKE}$. In $\mathsf{X\text{-}GA\text{-}PAKE}$, it is necessary to send/receive 2ℓ elements to prevent trivial attacks using twists, but in our scheme, by reducing ℓ elements of them to the number of common reference string (CRS), we can reduce the number of group actions per party to $4\ell + 2|\mathrm{CRS}|$ and the communication complexity per party to $\ell + |\mathrm{CRS}|$. In addition, we show the tight security in the one-round PAKE security model based on the same assumptions as in $\mathsf{X\text{-}GA\text{-}PAKE}$.

Keywords: password authenticated key exchange · isogenies · CSIDH · group actions

1 Introduction

Password authenticated key exchange (PAKE) is a cryptographic protocol to share a common session key using a shared low-entropy secret such as a human-memorable password among multiple parties through an unauthenticated channel such as the Internet. We focus on the 2-party PAKE. In PAKE, each party locally keeps a shared password, and in a key exchange session, each party generates an ephemeral public key (EPK) using ephemeral secret values such as randomness and sends the EPK to the other party. The session key is derived from these values and some key derivation functions like a hash function. Ordinary PAKE is intended to satisfy not only session key secrecy, but also offline/online dictionary attack resilience, and the provable security of PAKE schemes is formulated by indistinguishability-based security models such as the BPR model [7]

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 220–247, 2023.
https://doi.org/10.1007/978-3-031-35486-1_11

or the universal composability framework [9]. An offline dictionary attack is an attack in which the adversary obtains a message computed with a password, and attempts to find the password by locally testing guessed passwords. An online dictionary attack is an attack in which the adversary actually queries the system for guessed passwords and tries them until the system accepts them up to the maximum number of executions. For PAKE, it is required that the adversary is only possible to do online dictionary attacks because online dictionary attacks can be prevented by limiting the number of password trials, but offline dictionary attacks cannot be avoided by such an operational method.

In this paper, we focus on isogeny-based PAKE schemes.

1.1 Related Work

PAKE. Known PAKE schemes are classified in contexts of the Diffie-Hellman-based (DH-based) setting [4,14–16], the oblivious transfer-based (OT-based) setting [8] or the hash proof system-based (HPS-based) setting [12,13,18]. These strategies cannot be simply applied to isogeny-based PAKE due to the following limitations: (i) There is no known method yet to randomly hash the password into the set of supersingular elliptic curves. (ii) There is no natural ring structure on the set of elliptic curves that can be formed by multiplying two elliptic curves. (iii) It is not clear if the HPS from group actions [5] can be used as a building block for PAKE. Also, as shown in [2], OT-based PAKE is inefficient because the communication complexity and the number of group actions are quite large.

Isogeny-Based PAKE. Previously, there have been several studies to construct isogeny-based PAKE schemes. Isogeny-based PAKE schemes are classified into two settings: SIDH-based [17] and CSIDH-based [11].

Terada and Yoneyama [22] proposed a CSIDH-based PAKE scheme with the Encrypted Key Exchange [20] approach that the sent message is encrypted with a password as the secret key. Later, Azarderakhsh et al. [6] showed that the Terada-Yoneyama scheme is vulnerable to man-in-the-middle and offline dictionary attacks and modifying the scheme to be secure against man-in-the-middle attacks is hard due to the fact that the elliptic curves used in key exchange and encrypted with the password can be identified from random strings.

On the other hand, Taraskin et al. [21], and Soukharev and Hess [19] proposed SIDH-based PAKE schemes. These schemes have not been broken, but a polynomial-time key recovery attack on SIDH [10] was found by Castryck and Decru, which implies that SIDH-based PAKE is insecure now.

Recently, Abdalla et al. [2] extended the framework [5] of group actions for constructing isogeny-based cryptosystems such as CSIDH to capture the quadratic twist of elliptic curves, and proposed the first provably secure and tightly secure isogeny-based one-round PAKE scheme called X-GA-PAKE by a bit-by-bit approach. However, there are two problems in X-GA-PAKE. First, the security is proved in the model specialized for the sequential 2-pass or higher schemes, and therefore the security is not proved as a one-round scheme. Specifically, in the model, the adversary is only allowed to activate one of two parties in a session as the initiator because the sequence of SEND queries is fixed. It is desirable to allow the adversary to initialize both parties freely as one-round schemes. Next, there is a problem in efficiency because the number of group actions and the communication complexity per party are quite large due to sending a number of elements proportional to the password length. It is desirable to reduce these complexities.

1.2 Our Contribution

In this paper, we achieve a more compact isogeny-based PAKE scheme with tight security in the one-round PAKE security model than the existing scheme X-GA-PAKE. Our contribution is twofold.

- We propose a compact PAKE scheme. As shown in Table 1 in Sect. 4.4, in X-GA-PAKE, it is necessary to perform 5ℓ group actions and to send/receive 2ℓ elements to prevent trivial attacks using twists for a ℓ-bit password, but in our scheme, by reducing ℓ elements of them to the number of common reference string (CRS), we can reduce the number of group actions per party to $4\ell + 2|\mathrm{CRS}|$ and the communication complexity per party to $\ell + |\mathrm{CRS}|$.
- We prove that the proposed scheme is tightly secure in the one-round PAKE security model [1] based on the same assumptions as in X-GA-PAKE. Therefore, the security of our scheme is guaranteed even when parties send their messages simultaneously.

1.3 Key Technique

In X-GA-PAKE, double elements for each bit of the password are sent and received, and three shared values can be computed by the combination of these elements in order to provide resilience to an offline dictionary attack using twists. As a result, the number of group actions per user is 5ℓ times, and the communication complexity per user is 2ℓ group elements. Here, there are two essential points to prevent the attack. The first is to send two types of messages according to password bits. The second is to compute three shared values, including the normal DH-style key exchange and the key exchange with message crossing, i.e., it corresponds to computing g^{ab}, $g^{a'b}$, and $g^{ab'}$ in the classical DH key exchange between user A and user B. Thus, we propose a technique to achieve these points in which ephemeral CRSs are generated and sent to each other instead of doubling the message. Then, in the computation of the shared values using such messages, the computation of shared values is blanched based on the password. In this way, we reduce the number of group actions per user can be reduced to $4\ell+2$ times and the communication complexity per user to $\ell+2$ group elements. For more details, please see Sect. 4.1.

2 Preliminaries

2.1 Notation

- For integers m and n, $[m,n]$ denotes the set $\{m, m+1, \ldots, n\}$ where $m < n$. In the case of $m = 1$, we denote it as $[n]$.
- For a set X, $x \in_R X$ denotes that the element x is sampled uniformly and randomly from the set X.
- $y \leftarrow \mathcal{A}(x_1, \ldots, x_n)$ denotes that probabilistic polynomial time (PPT) adversary $\mathcal{A}$ takes $(x_1, \ldots, x_n)$ as input and outputs y.
- $\mathcal{A}^{\mathcal{O}}$ denotes that $\mathcal{A}$ has access to oracle $\mathcal{O}$.

2.2 Effective Group Actions (with Twists)

Here, we recall the definition of effective group actions with twists from [2], which extends the framework [5] to build cryptographic primitives relying on isogeny-based assumptions such as CSIDH to one with twists capability.

Definition 1 (Group Action). *Let $(\mathcal{G}, \cdot)$ be a group with identity element $id \in \mathcal{G}$, and $\mathcal{X}$ be a set. We say that $(\mathcal{G}, \mathcal{X}, \star)$ for a map*

$$\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$$

is a group action if it satisfies the following properties:

1. *Identity: $id \star x = x$ for all $x \in \mathcal{X}$.*
2. *Compatibility: $(g \cdot h) \star x = g \star (h \star x)$ for all $g, h \in \mathcal{G}$ and $x \in \mathcal{X}$.*

In this paper, we consider group actions with finite commutative groups $\mathcal{G}$. In addition, we assume that the group action is regular which means that there exists exactly one $g \in \mathcal{G}$ satisfying $y = g \star x$ for any $x, y \in \mathcal{X}$.

Definition 2 (Effective Group Action). *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action satisfying the following properties:*

1. *The group $\mathcal{G}$ is finite and there exist efficient (PPT) algorithms for membership and equality testing, (random) sampling, group operation, and inversion.*
2. *The set $\mathcal{X}$ is finite and there exist efficient algorithms for membership testing and computing a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ (called the origin) with known representation.*
4. *There exists an efficient algorithm to evaluate the group action, i.e., to compute $g \star x$ given g and x.*

We say that $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ is an effective group action (EGA) *for above $(\mathcal{G}, \mathcal{X}, \star)$ and the origin $\tilde{x} \in \mathcal{X}$.*

Definition 3 (EGA with Twists). *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action satisfying the following properties:*

1. *$(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ is* EGA *for the origin $\tilde{x} \in \mathcal{X}$.*
2. *There exists an efficient algorithm that given $x = g \star \tilde{x} \in \mathcal{X}$ computes $x^t = g^{-1} \star \tilde{x}$.*

We say that $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ is an effective group action with twists (EGAT) *for above $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$.*

Remark 1. The instantiation of the group actions described above is provided by isogeny-based group actions. Specifically, we can define group actions $Cl(\mathcal{O}) \times \mathcal{E}ll_k(\mathcal{O}) \to \mathcal{E}ll_k(\mathcal{O})$, where $Cl(\mathcal{O})$ is the ideal class group and $\mathcal{E}ll_k(\mathcal{O})$ is the set of elliptic curves. Also, for any curve $E_a = [\mathfrak{a}] \star E$ which is the form of $E_a : y^2 = x^3 + ax^2 + x$, the quadratic twists is computed as $(E_a)^t = E_{-a}$ easily and satisfies the property $(E_a)^t = [\mathfrak{a}]^{-1} \star E$.

2.3 Assumptions of Group Actions

Here, we recall the three computational problems Group Action Strong Computational DH (GA-StCDH), Square-Inverse GA-StCDH (SqInv-GA-StCDH) [2], and Double Simultaneous GA-StCDH (DSim-GA-StCDH) [2], to be used in the proof of our scheme.

As a notation, $\mathsf{GA\text{-}CDH}_x(y_0, y_1)$ is the computational oracle that computes $g_0 \star y_1$ and $\mathsf{GA\text{-}DDH}_x(y_0, y_1, z)$ is the decision oracle that verifies $\mathsf{GA\text{-}CDH}_x(y_0, y_1) = z$, where $g_0 \in \mathcal{G}$ such that $y_0 = g_0 \star x$. If $\mathsf{GA\text{-}CDH}_x(y_0, y_1) = z$, it outputs 1, otherwise 0. For GA-CDH and GA-DDH, we omit the index x if $x = \tilde{x}$ (e.g., $\mathsf{GA\text{-}CDH}_{\tilde{x}}(y_0, y_1) = \mathsf{GA\text{-}CDH}(y_0, y_1)$).

Definition 4 (GA-StCDH). *For any PPT adversary $\mathcal{A}$, the* GA-StCDH *problem requires to compute the set element $(g \cdot h) \star \tilde{x}$ based on $(g \star \tilde{x}, h \star \tilde{x}) \in \mathcal{X}^2$ given as input. For an group action* EGAT*, the advantage of $\mathcal{A}$ is shown as follows:*

$$Adv_{\mathsf{EGAT}}^{\mathsf{GA\text{-}StCDH}}(\mathcal{A}) := \Pr[(g \cdot h) \star \tilde{x} \leftarrow \mathcal{A}^{\mathcal{O}}(g \star \tilde{x}, h \star \tilde{x})],$$

where $(g, h) \in_R \mathcal{G}^2$ and $\mathcal{O}$ is the decision oracle $\mathsf{GA\text{-}DDH}(g \star \tilde{x}, \cdot, \cdot)$.

Definition 5 (SqInv-GA-StCDH). *For any PPT adversary $\mathcal{A}$, the* SqInv-GA-StCDH *problem requires to find a tuple $(y, y_0, y_1) \in \mathcal{X}^3$ such that $y_0 = g^2 \star y$ and $y_1 = g^{-1} \star y$, based on $x = g \star \tilde{x}$ given as input. For an group action* EGAT*, the advantage of $\mathcal{A}$ is shown as follows:*

$$\begin{aligned} Adv_{\mathsf{EGAT}}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}(\mathcal{A}) := \Pr[(&y, y_0 = \mathsf{GA\text{-}CDH}_{x^t}(x, y), \\ &y_1 = \mathsf{GA\text{-}CDH}(x^t, y)) \leftarrow \mathcal{A}^{\mathcal{O}}(g \star \tilde{x})], \end{aligned}$$

where $\mathcal{O}$ is the decision oracle $\{\mathsf{GA\text{-}DDH}_{x^t}(x, \cdot, \cdot), \mathsf{GA\text{-}DDH}(x, \cdot, \cdot)\}$.

Remark 2. $\mathcal{A}$ can choose y only based on known elements based on $\tilde{x}$, its input x or x^t.

The $\mathsf{SqInv\text{-}GA\text{-}StCDH}$ problem is also hard against an adversary that has the ability to compute the twists. Specifically, if $\mathcal{A}$ chooses $y = a \star \tilde{x}$ for some $a \in \mathcal{G}$, then it can compute $y_1 = a \star x^t$, but not $y_0 = (a \cdot g^2) \star \tilde{x}$. If $\mathcal{A}$ chooses $y = a \star x$, then it can compute $y_1 = a \star \tilde{x}$, but not compute $y_0 = (a \cdot g^3) \star \tilde{x}$. Also, if $\mathcal{A}$ chooses $y = a \star x^t$, it can compute $y_0 = a \star x$, but not compute $y_1 = (a \cdot g^{-2}) \star \tilde{x}$.

Definition 6 ($\mathsf{DSim\text{-}GA\text{-}StCDH}$). *For any PPT adversary $\mathcal{A}$, the* $\mathsf{DSim\text{-}GA\text{-}StCDH}$ *problem requires to find a tuple $(y, y_0, y_1, y_2, y_3) \in \mathcal{X}^5$ such that*

$$\begin{aligned}(y_0,&y_1, y_2, y_3) = \\ &(g_0^{-1} \cdot h_0 \star y, g_0^{-1} \cdot h_1 \star y, g_1^{-1} \cdot h_0 \star y, g_1^{-1} \cdot h_1 \star y),\end{aligned}$$

based on $(x_0, x_1, w_0, w_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, h_0 \star \tilde{x}, h_1 \star \tilde{x}) \in \mathcal{X}^4$ given as input. For an group action EGAT*, the advantage of $\mathcal{A}$ is shown as follows:*

$$\begin{aligned}Adv_{\mathsf{EGAT}}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}(\mathcal{A}) := |\Pr[(&y_0 = \mathsf{GA\text{-}CDH}_{x_0}(w_0, y), \\ &y_1 = \mathsf{GA\text{-}CDH}_{x_0}(w_1, y), y_2 = \mathsf{GA\text{-}CDH}_{x_1}(w_0, y), \\ &y_3 = \mathsf{GA\text{-}CDH}_{x_1}(w_1, y)) \leftarrow \mathcal{A}^{\mathcal{O}}(x_0, x_1, w_0, w_1)]|,\end{aligned}$$

where $\mathcal{O}$ is the decision oracle $\{\mathsf{GA\text{-}DDH}_{x_j}(w_k, y)\}_{j,k\in\{0,1\}}$.

The $\mathsf{DSim\text{-}GA\text{-}StCDH}$ problem can be reduced to the $\mathsf{SqInv\text{-}GA\text{-}StCDH}$ problem as in [2].

3 Security Model for One-Round PAKE

In this section, we recall the security model [1] which is an extension of the BPR model [7] to multiple test queries [3]. The security model is indistinguishability-based, and we use the game-based pseudocode in our proof as in [2]. In addition, the model is different from the model used in [2] in the point that it is possible to capture the one-round schemes. Specifically, it has only one SEND query, allowing the adversary to query freely without being restricted to an initialized party.

3.1 System Model and Adversarial Capacity

PAKE Setting. Parties are modeled as a PPT Turing machine. Each party is activated by receiving an initialization message and returns a message defined by the protocol. PAKE is executed using a shared password $\mathsf{pw_{US}}$ between the two parties U and S. Let $\mathcal{U}$ be the namespaces of parties, and we assume that U and S are disjoint where $(\mathsf{U}, \mathsf{S}) \in \mathcal{U}$. Also, let a party P be $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$ and an intended peer $\bar{\mathsf{P}}$ be $\bar{\mathsf{P}} \in (\mathsf{U}, \mathsf{S}) \backslash \mathsf{P}$. Each party P has multiple instances π_{P}^i and each instance has its own state. Passwords are bit strings of length ℓ and we define the password space as $\mathcal{PW} \subsetneq \{0, 1\}^\ell$.

Protocol Instances. Let EPK_{P} be an ephemeral public key of P. An instance π_{P}^{t} is activated by an incoming message of the form $(\mathsf{P}, t, \bar{\mathsf{P}}, msg)$, where $msg = \perp$. The instance π_{P}^{t} has a tuple as follows:

- e: It stores the ephemeral secret values generated in the protocol execution.
- tr: It stores the trace of the instance in the protocol execution, i.e., the messages generated by the party and received from the intended peer are stored. The trace $\pi_{\mathsf{P}}^{t}.\mathsf{tr}$ is the form of $(\mathsf{P}, \bar{\mathsf{P}}, EPK_{\mathsf{P}}, \perp)$ or $(\mathsf{P}, \bar{\mathsf{P}}, EPK_{\mathsf{P}}, EPK_{\bar{\mathsf{P}}})$.
- SK: It stores the accepted session key in the protocol execution.
- acc: It stores the status of protocol execution. As long as the instance did not receive the last message, $acc = \perp$. If the instance π_{P}^{t} computes the session key SK in the protocol execution, $\pi_{\mathsf{P}}^{t}.acc$ stores "**true**".

To access individual components of the state, we write $\pi_{\mathsf{P}}^{t}.\{\mathsf{e}, \mathsf{tr}, SK, acc\}$.

Partnering. Partnering is defined by matching conversations. Especially, the instances $\pi_{\mathsf{U}}^{t_0}$ and $\pi_{\mathsf{S}}^{t_1}$ are partnered if

$$\pi_{\mathsf{U}}^{t_0}.acc = \pi_{\mathsf{S}}^{t_1}.acc = \textbf{true} \text{ and } \pi_{\mathsf{U}}^{t_0}.\mathsf{tr} = \pi_{\mathsf{S}}^{t_1}.\mathsf{tr}$$

We define a partner predicate $\mathsf{Partner}(\pi_{\mathsf{U}}^{t_0}, \pi_{\mathsf{S}}^{t_1})$ which outputs 1 if the two instances $\pi_{\mathsf{U}}^{t_0}$ and $\pi_{\mathsf{S}}^{t_1}$ are partnered, otherwise 0.

Adversary. The adversary $\mathcal{A}$ is modeled as a PPT turing machine, which takes *params* as input and has oracle access to parties. $\mathcal{A}$ controls all communications among users including the session activation. $\mathcal{A}$ can interfere in party P and $\bar{\mathsf{P}}$ to execute a specific action using the following adversarial queries.

- $\mathsf{EXECUTE}(\mathsf{U}, t_0, \mathsf{S}, t_1)$: This query is to compute a protocol execution between an instance $\pi_{\mathsf{U}}^{t_0}$ and an instance $\pi_{\mathsf{S}}^{t_1}$. The query models the behavior of passive adversaries.
- $\mathsf{SEND}(\mathsf{P}, t, msg)$: The adversary sends an arbitrary message msg to the instance π_{P}^{t}. The msg has one of the following forms : $\perp$ or $EPK_{\bar{\mathsf{P}}}$. π_{P}^{t} executes the protocol according to the given message and returns the response to $\mathcal{A}$. The query models the behavior of active adversaries.
- $\mathsf{CORRUPT}(\mathsf{U}, \mathsf{S})$: The adversary obtains the shared password $\mathsf{pw}_{\mathsf{US}}$ of U and S.
- $\mathsf{REVEAL}(\mathsf{P}, t)$: The adversary obtains the session key of the instance π_{P}^{t}.

3.2 PAKE Security

For defining PAKE security, we need the notion of freshness.

Freshness. Let $\pi_{\bar{\mathsf{P}}}^{t_1}$ be a partner instance of $\pi_{\mathsf{P}}^{t_0}$. During the security experiment, we register if an instance is fresh to prevent trivial attacks. Therefore, we define a freshness predicate $\mathsf{Fresh}(\mathsf{P}, t_0)$. We say that the instance $\pi_{\mathsf{P}}^{t_0}$ is fresh as $\mathsf{Fresh}(\mathsf{P}, t_0) = \textbf{true}$ if the following conditions hold:

1. $\pi_{\mathsf{P}}^{t_0}$ accepted.
2. $\pi_{\mathsf{P}}^{t_0}$ was not queried to $\mathsf{TEST}(\mathsf{P}, t_0)$ or $\mathsf{REVEAL}(\mathsf{P}, t_0)$ before.
3. $\pi_{\bar{\mathsf{P}}}^{t_1}$ was not queried to $\mathsf{TEST}(\bar{\mathsf{P}}, t_1)$ or $\mathsf{REVEAL}(\bar{\mathsf{P}}, t_1)$ before.
4. At least one of the following conditions holds:
   (a) $\pi_{\mathsf{P}}^{t_0}$ accepted during a query to $\mathsf{EXECUTE}(\pi_{\mathsf{P}}^{t_0}, \pi_{\bar{\mathsf{P}}}^{t_1})$.
   (b) A unique fresh partner instance $\pi_{\bar{\mathsf{P}}}^{t_1}$ exists.
   (c) No partner exists and $\mathsf{CORRUPT}(\bar{\mathsf{P}}, \cdot)$ was not queried.

The goal of the adversary $\mathcal{A}$ in the security experiment is to distinguish the true session key from a random key. $\mathcal{A}$ makes any sequence of the above queries. $\mathcal{A}$ makes the following query during the experiment.

- $\mathsf{TEST}(\mathsf{P}, t_0)$: This query is a challenge query. Here, $\pi_{\mathsf{P}}^{t_0}$ must be fresh (i.e., $\mathsf{Fresh}(\mathsf{P}, t_0) = \textbf{true}$). Depending on the challenge bit $\beta \in_R \{0, 1\}$ chosen before the experiment begins, it returns the challenge key. If $\beta = 0$, then it returns the session key $\pi_{\mathsf{P}}^{t_0}.SK$. Otherwise, it returns a random key $SK \in_R \mathcal{SK}$, where $\mathcal{SK}$ is the session key space. If TEST is queried for an instance $\pi_{\mathsf{P}}^{t_0}$, we mark an instance as tested by $\pi_{\mathsf{P}}^{t_0}.test = \textbf{true}$.

The adversary $\mathcal{A}$ obtains either the session key of $\pi_{\mathsf{P}}^{t_0}$ or a random key with probability 1/2 respectively. After issuing the Test query, the experiment continues until $\mathcal{A}$ outputs β' as a result of guessing whether the received key is random or not. If the guess of $\mathcal{A}$ is correct (i.e., $\beta = \beta'$), then $\mathcal{A}$ wins the experiment.

Definition 7 (PAKE Security). *The advantage of the adversary $\mathcal{A}$ in the above experiment with the PAKE scheme is defined as follows.*

$$Adv^{\mathsf{PAKE}}(\mathcal{A}) := |\Pr[\mathcal{A}\ win] - 1/2|$$

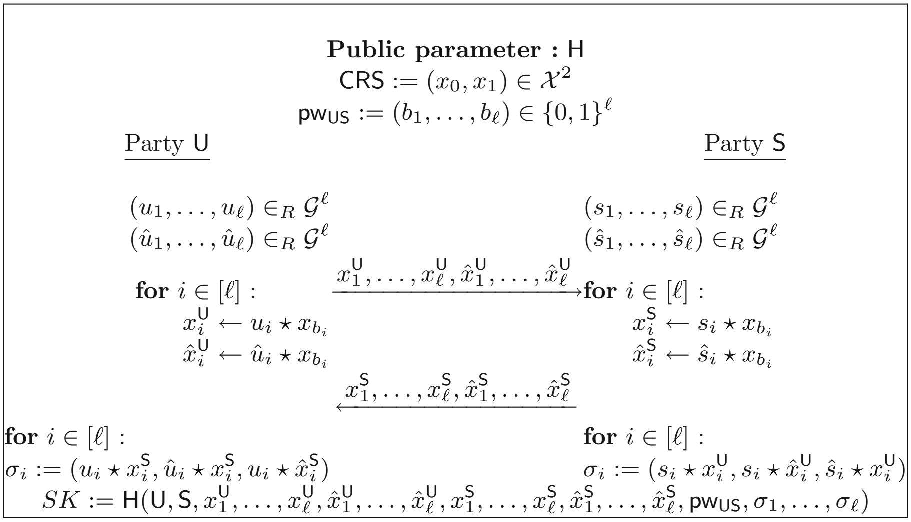
PAKE is considered secure if an online dictionary attack is the best threat by the adversary. More specifically, this means that the advantage of the adversary should be negligible close to $q_s/|\mathcal{PW}|$ when passwords are chosen uniformly and independently from $\mathcal{PW}$, where q_s is the maximum number of SEND queries made by the adversary.

4 Our PAKE Protocol: $\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$

In this section, we propose a one-round PAKE scheme ($\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$) in the random oracle model (ROM). $\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$ is secure under the hardness of the computational problems $\mathsf{GA\text{-}StCDH}$ and $\mathsf{SqInv\text{-}GA\text{-}StCDH}$ in the one-round security model. The protocol of $\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$ is shown in Fig. 2.

4.1 Construction Idea

As discussed in Sect. 1.3, we try to reduce the communication complexity and the number of group actions of $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ shown in Fig. 1. In $\mathsf{GA\text{-}PAKE}$ in [2],



Public parameter : H

$\mathsf{CRS} := (x_0, x_1) \in \mathcal{X}^2$

$\mathsf{pw}_{\mathsf{US}} := (b_1, \ldots, b_\ell) \in \{0,1\}^\ell$

Party U | Party S

$(u_1, \ldots, u_\ell) \in_R \mathcal{G}^\ell$ | $(s_1, \ldots, s_\ell) \in_R \mathcal{G}^\ell$

$(\hat{u}_1, \ldots, \hat{u}_\ell) \in_R \mathcal{G}^\ell$ | $(\hat{s}_1, \ldots, \hat{s}_\ell) \in_R \mathcal{G}^\ell$

for $i \in [\ell]$: $x_i^{\mathsf{U}} \leftarrow u_i \star x_{b_i}$, $\hat{x}_i^{\mathsf{U}} \leftarrow \hat{u}_i \star x_{b_i}$ | **for** $i \in [\ell]$: $x_i^{\mathsf{S}} \leftarrow s_i \star x_{b_i}$, $\hat{x}_i^{\mathsf{S}} \leftarrow \hat{s}_i \star x_{b_i}$

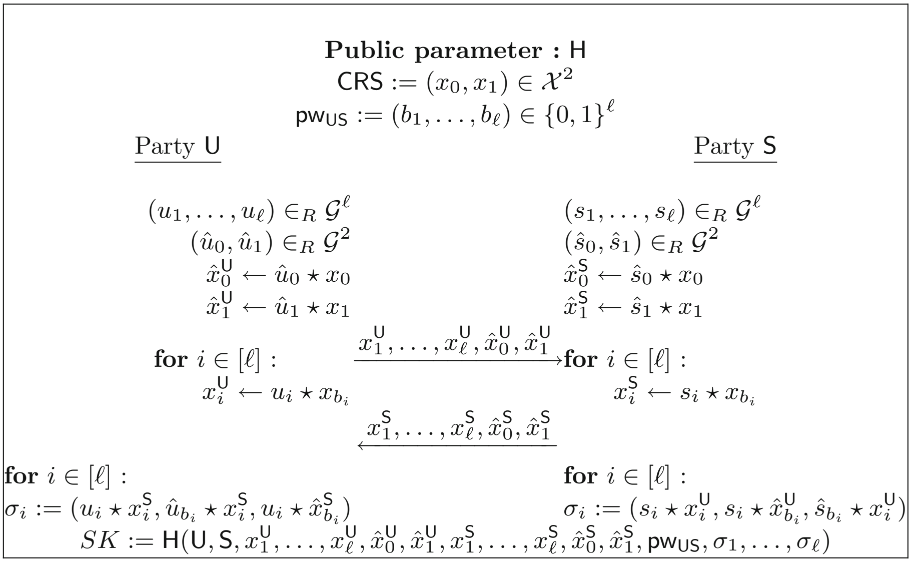
$\xrightarrow{x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}, \ldots, \hat{x}_\ell^{\mathsf{U}}}$

$\xleftarrow{x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}, \ldots, \hat{x}_\ell^{\mathsf{S}}}$

for $i \in [\ell]$: $\sigma_i := (u_i \star x_i^{\mathsf{S}}, \hat{u}_i \star x_i^{\mathsf{S}}, u_i \star \hat{x}_i^{\mathsf{S}})$ | **for** $i \in [\ell]$: $\sigma_i := (s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_i^{\mathsf{U}}, \hat{s}_i \star x_i^{\mathsf{U}})$

$SK := \mathsf{H}(\mathsf{U}, \mathsf{S}, x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}, \ldots, \hat{x}_\ell^{\mathsf{U}}, x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}, \ldots, \hat{x}_\ell^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma_1, \ldots, \sigma_\ell)$

Fig. 1. $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ [2]



Public parameter : H

$\mathsf{CRS} := (x_0, x_1) \in \mathcal{X}^2$

$\mathsf{pw}_{\mathsf{US}} := (b_1, \ldots, b_\ell) \in \{0,1\}^\ell$

Party U | Party S

$(u_1, \ldots, u_\ell) \in_R \mathcal{G}^\ell$ | $(s_1, \ldots, s_\ell) \in_R \mathcal{G}^\ell$

$(\hat{u}_0, \hat{u}_1) \in_R \mathcal{G}^2$ | $(\hat{s}_0, \hat{s}_1) \in_R \mathcal{G}^2$

$\hat{x}_0^{\mathsf{U}} \leftarrow \hat{u}_0 \star x_0$ | $\hat{x}_0^{\mathsf{S}} \leftarrow \hat{s}_0 \star x_0$

$\hat{x}_1^{\mathsf{U}} \leftarrow \hat{u}_1 \star x_1$ | $\hat{x}_1^{\mathsf{S}} \leftarrow \hat{s}_1 \star x_1$

for $i \in [\ell]$: $x_i^{\mathsf{U}} \leftarrow u_i \star x_{b_i}$ | **for** $i \in [\ell]$: $x_i^{\mathsf{S}} \leftarrow s_i \star x_{b_i}$

$\xrightarrow{x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}}$

$\xleftarrow{x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}}$

for $i \in [\ell]$: $\sigma_i := (u_i \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \star x_i^{\mathsf{S}}, u_i \star \hat{x}_{b_i}^{\mathsf{S}})$ | **for** $i \in [\ell]$: $\sigma_i := (s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_{b_i}^{\mathsf{U}}, \hat{s}_{b_i} \star x_i^{\mathsf{U}})$

$SK := \mathsf{H}(\mathsf{U}, \mathsf{S}, x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}, x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma_1, \ldots, \sigma_\ell)$

Fig. 2. $\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$

it allows the adversary an offline dictionary attack using twists trivially. Concretely, the adversary performs group actions with twisted opponent's messages instead of group actions with CRS. As a result, the opponent's ephemeral secret values are canceled out when it computes the shared values; thus, an offline dictionary attack can be possible. Therefore, in X-GA-PAKE, sending/receiving double elements and making them compute three shared values prevents the attack.

Carefully investigating the proof of X-GA-PAKE, there are two essential points to prevent attacks using twists. The first is to send two types of messages according to password bits. The second is to compute three shared values, including the normal DH-style key exchange and the key exchange with message crossing. To achieve these points, we discover that sending messages depending on the password length ℓ twice is not necessary. Thus, we propose a technique in which ephemeral CRSs are generated and sent to each other instead of doubling the message. Then, in the computation of the shared values using such messages, the computation of shared values is blanched based on the password. In this way, the attack using twists can be prevented by changing the timing of branching by the password.

In the proof of CrsX-GA-PAKE, by the definition of the adversary, we need to consider passive adversaries and active adversaries. For a passive adversary, the adversary cannot know the password and ephemeral secret values; thus, it can be reduced the hardness of GA-StCDH problem in which let one type of value be output. For an active adversary, the adversary can create one party's ephemeral secret values on its own, and an attack using twists is possible for one type of shared value. Thus, the hardness of DSim-GA-StCDH problem can be reduced in which let two types of values be output simultaneously.

4.2 Protocol

Public Parameters and CRS: Let κ be a security parameter, $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ be a hash function, (x_0, x_1) satisfying $x_0 = g_0 \star \tilde{x}$ and $x_1 = g_1 \star \tilde{x}$ be elliptic curves with group elements g_0 and g_1. These are provided as part of the public parameters and CRS.

Key Exchange: Let U and S be the parties that execute the protocol with a shared password $\mathsf{pw}_{\mathsf{US}} \in \{0,1\}^\ell$.

- U chooses ephemeral secret values $(u_1, \ldots, u_\ell) \in_R \mathcal{G}^\ell$ and $(\hat{u}_0, \hat{u}_1) \in_R \mathcal{G}^2$, and computes $\hat{x}_0^{\mathsf{U}} \leftarrow \hat{u}_0 \star x_0$, $\hat{x}_1^{\mathsf{U}} \leftarrow \hat{u}_1 \star x_1$, and $x_i^{\mathsf{U}} \leftarrow u_i \star x_{b_i}$ for $i \in [\ell]$. Then, U sends $(x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}})$ to S. S executes in the same way.
- Upon receiving $(x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}})$, U computes $\sigma_i := (u_i \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \star x_i^{\mathsf{S}}, u_i \star \hat{x}_{b_i}^{\mathsf{S}})$ for $i \in [\ell]$ and the session key $SK = \mathsf{H}(\mathsf{U}, \mathsf{S}, x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}, x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma_1, \ldots, \sigma_\ell)$.
- Upon receiving $(x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}})$, S computes $\sigma_i := (s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_{b_i}^{\mathsf{U}}, \hat{s}_{b_i} \star x_i^{\mathsf{U}})$ for $i \in [\ell]$ and the session key $SK = \mathsf{H}(\mathsf{U}, \mathsf{S}, x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}, x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma_1, \ldots, \sigma_\ell)$.

4.3 Security

We show the security of the $\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$ for EGAT.

Theorem 1 (Security of $\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$). *For any PPT adversary $\mathcal{A}$ against $\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$ that issues at most q_e $\mathsf{EXECUTE}$ queries and q_s SEND queries where H is modeled as a random oracle, there exist a PPT adversary $\mathcal{B}_1$ against $\mathsf{GA\text{-}StCDH}$ and a PPT adversary $\mathcal{B}_2$ against $\mathsf{SqInv\text{-}GA\text{-}StCDH}$ such that*

$$
\begin{aligned}
&Adv^{\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A})\\
&\le Adv^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + Adv^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2) + \frac{q_s}{|\mathcal{PW}|} + \frac{(q_s+q_e)^2}{|\mathcal{G}|^{\ell+2}}.
\end{aligned}
$$

We show the proof of Theorem 1 in Appendix A.

Remark 3. The reduction cost of our proof is the same as the proof of $\mathsf{X\text{-}GA\text{-}PAKE}$. $Adv^{\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A})$ is tightly reduced to $Adv^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + Adv^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2)$. $\frac{q_s}{|\mathcal{PW}|}$ is inevitable in the PAKE setting due to online dictionary attacks and $\frac{(q_s+q_e)^2}{|\mathcal{G}|^{\ell+2}}$ is negligible.

Sketch of Proof for Active Adversary. We consider the case that an active adversary $\mathcal{A}$ impersonates a user and attacks, including the twist attack with a SEND query, i.e., $\mathcal{A}$ can determine the EPK for one of the users by own. In the proof, we reduce the advantage of $\mathcal{A}$ to the security of $\mathsf{DSim\text{-}GA\text{-}StCDH}$. Initially, the adversary $\mathcal{B}_2$ against $\mathsf{DSim\text{-}GA\text{-}StCDH}$ receives $(x_0, x_1, w_0, w_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, h_0 \star \tilde{x}, h_1 \star \tilde{x}) \in \mathcal{X}^4$. First, $\mathcal{B}_2$ embeds x_0 and x_1 in the input of the $\mathsf{DSim\text{-}GA\text{-}StCDH}$ problem to the CRS of the PAKE game. Second, $\mathcal{B}_2$ generates messages for user U as follows:

$$
\begin{aligned}
x_i^{\mathsf{U}} &= u_i \star w_0 = (u_i \cdot h_0 \cdot g_0^{-1}) \star x_0 = (u_i \cdot h_0 \cdot g_1^{-1}) \star x_1,\\
\hat{x}_0^{\mathsf{U}} &= \hat{u}_0 \star w_1 = (\hat{u}_0 \cdot h_1 \cdot g_0^{-1}) \star x_0, \hat{x}_1^{\mathsf{U}} = \hat{u}_1 \star w_1 = (\hat{u}_1 \cdot h_1 \cdot g_1^{-1}) \star x_1,
\end{aligned}
$$

where the value in brackets is the value used for the DH computation in the actual protocol. For the other user S, $\mathcal{B}_2$ generates messages analogously using the secret group elements s_i, $\hat{s}_0$ and $\hat{s}_1$. Finally, $\mathcal{B}_2$ answers the problem as follows:

- **Case 1:** If $\mathcal{A}$ impersonates U, since $\mathcal{B}_2$ does not know S's ephemeral secrets, $\mathcal{B}_2$ outputs the answer using u_i, $\hat{u}_0$, $\hat{u}_1$ and the shared values of the instance.
- **Case 2:** If $\mathcal{A}$ impersonates S, since $\mathcal{B}_2$ does not know U's ephemeral secrets, $\mathcal{B}_2$ outputs the answer using s_i, $\hat{s}_0$, $\hat{s}_1$ and the shared values of the instance.

We simulate in this way to prove that $\mathcal{A}$ cannot simultaneously derive the correct shared values.

4.4 Comparison with Existing Scheme

We show the comparison of the efficiency with $\mathsf{X\text{-}GA\text{-}PAKE}$ in Table 1.

In $\mathsf{X\text{-}GA\text{-}PAKE}$, for the password length ℓ, the number of group actions per party is 5ℓ, and the number of sent elements per party is 2ℓ. If we use the

Table 1. Comparison between X-GA-PAKE and our scheme

Protocol	Assumption	# of rounds	Proved model	# of CRS	# of group action	# of elements
X-GA-PAKE$_\ell$ [2]	SqInv-GA-StCDH	1	2-move	2	5ℓ 640	2ℓ 256
X-GA-PAKE$^t_{\ell,N}$ [2]	SqInv-GA-StCDH	1	2-move	2^{N-1}	$5\ell/N$ 160	$2\ell/N$ 64
CrsX-GA-PAKE$_\ell$	SqInv-GA-StCDH	1	1-round	2	$4\ell+2$ 512	$\ell+2$ 130
CrsX-GA-PAKE$^t_{\ell,N}$	SqInv-GA-StCDH	1	1-round	2^{N-1}	$4\ell/N+2^{N-1}$ 130	$\ell/N+2^{N-1}$ 40

For X-GA-PAKE$^t_{\ell,N}$ and CrsX-GA-PAKE$^t_{\ell,N}$, we apply the technique of increasing the number of CRS to 2^N and dividing the length of passwords into ℓ/N, and the technique [2] of using twists in the setup, which reduce the number of CRS by half. For concreteness, expected values for a 128-bit implementation for $N = 4$ are also given.

technique [2] of using twists in the setup, in our scheme, the number of group actions per party is $4\ell + 2|\mathrm{CRS}|$ and the number of sent elements per party is $\ell + 2|\mathrm{CRS}|$. The optimization technique of the block-by-block approach can be applied to our scheme. Though when the block size is larger, the number of group actions and sent elements of X-GA-PAKE can be smaller than our scheme, a large number of CRS are necessary. In addition, X-GA-PAKE is the one-round scheme, but there is no security proof for the one-round execution since the security model is restricted to sequential executions. However, our scheme has tight security in the one-round PAKE security model based on the same assumptions as in X-GA-PAKE.

5 Conclusion

In this paper, we proposed an efficient one-round PAKE scheme that reduces the number of group actions and the communication complexity compared to X-GA-PAKE. However, our scheme still depends on the password length ℓ for the number of group actions and the communication complexity. It is a remaining problem to construct PAKE schemes that do not depend on ℓ for these values without compromising security.

A Proof of Theorem 1

Proof. Let κ be a security parameter. In the PAKE security game, the maximum number of $\mathsf{EXECUTE}$ query is q_e and the maximum number of SEND query is q_s. Let adversary $\mathcal{A}$ be a PPT adversary in κ against $\mathsf{CrsX\text{-}GA\text{-}PAKE}_\ell$, and we construct the adversary $\mathcal{B}_1$ against $\mathsf{GA\text{-}StCDH}$ or the adversary $\mathcal{B}_2$ against $\mathsf{SqInv\text{-}GA\text{-}StCDH}$ using $\mathcal{A}$ performs the PAKE security game.

We change the interface of oracle queries and the computation of the session key. These instances are gradually changed over seven hybrid experiments in Figs. 3, 4, 6, 7, 8 and 9 depending on specific subcases. In the last hybrid experiment, the session key in the test session does not contain information of the bit b and the advantage of $\mathcal{A}$ is negligible close to $q_s/|\mathcal{PW}|$. Thus, the adversary clearly only outputs a random guess. We denote these hybrid experiments by $\mathbf{H}_0, \ldots, \mathbf{H}_6$, and the advantage of the adversary $\mathcal{A}$ when participating in experiment $\mathbf{H}_i$ by $Adv(\mathcal{A}, \mathbf{H}_i)$.

Each game keeps the list L_{H} that keep the queries and answers of H oracle, the list L_{COR} that holds the answers of $\mathsf{CORRUPT}$, and the list L_e that holds the session state and the answers of $\mathsf{EXECUTE}$.

Hybrid Experiment $\mathbf{H_0}$: This experiment denotes the real experiment for PAKE security and in this experiment, the environment for $\mathcal{A}$ is as defined in the protocol. Thus, $Adv(\mathcal{A}, \mathbf{H_0})$ is the same as the advantage of the real experiment.

Hybrid Experiment $\mathbf{H_1}$: In the experiment $\mathbf{H_1}$, if the trace matches another accepted instance, we set the flag $\mathbf{bad}_{coll}$. If the flag is set, SEND query returns $\perp$ (line 64). $\mathsf{EXECUTE}$ query also returns $\perp$ (line 28) if the trace computed in the query collides with one of a previously accepted instance. From the difference lemma, $|Adv(\mathcal{A}, \mathbf{H_1}) - Adv(\mathcal{A}, \mathbf{H_0})| \leq \Pr[\mathbf{bad}_{coll}]$. If $\mathbf{bad}_{coll}$ does not set, each instance is unique and there is at most one partner instance.

The trace of an instance π_{P}^t consists of $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}} = (x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}), \hat{x}^{\mathsf{U}} = (\hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}),\ x^{\mathsf{S}} = (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}), \hat{x}^{\mathsf{S}} = (\hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}))$. However, the message for either $(x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$ or $(x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ was generated by a SEND or $\mathsf{EXECUTE}$ query. Thus, $\mathbf{bad}_{coll}$ can only occur if all those $\ell+2$ set elements collide with all $\ell+2$ set elements of another instance. The probability that this happens for two sessions is $|\mathcal{G}^{-(\ell+2)}|$ and the number of instances is $q_e + q_s$, and thus $|Adv(\mathcal{A}, \mathbf{H_1}) - Adv(\mathcal{A}, \mathbf{H_0})| \leq \Pr[\mathbf{bad}_{coll}] \leq \frac{(q_e+q_s)^2}{|\mathcal{G}^{\ell+2}|}$.

Hybrid Experiment $\mathbf{H_2}$: In the experiment H_2, We explicitly evaluate the freshness of instances. For each instance π_{P}^t, we add a variable $\pi_{\mathsf{P}}^t.\mathsf{fr}$ which is updated during the experiment.

```
Experiments $\mathbf{H_0} - \mathsf{H}_4$
$(g_0, g_1) \in_R \mathcal{G}^2$
$(x_0, x_1) := (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
$(L_{\mathsf{H}}, L_{\mathsf{COR}}) := (\emptyset, \emptyset)$
$\mathbf{bad}_{coll} := \mathbf{false}$                                          $/\!/\mathbf{H_1}$-$\mathbf{H_4}$
$\beta \in_R \{0, 1\}$
$\mathbf{for}\ (\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}:$
    $\mathsf{pw}_{\mathsf{US}} \in_R \mathcal{PW}$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}}(x_0, x_1)$
$\mathbf{return}\ 1\ \mathbf{or}\ \perp$

$\mathsf{EXECUTE}(\mathsf{U}, t_0, \mathsf{S}, t_1)$
$\mathbf{if}\ \pi_{\mathsf{U}}^{t_0} \neq \perp\ \mathbf{or}\ \pi_{\mathsf{S}}^{t_1} \neq \perp$
    $\mathbf{return}\ \perp$
$(b_1, \ldots, b_\ell) := \mathsf{pw}_{\mathsf{US}}$                              $/\!/\mathbf{H_0}$-$\mathbf{H_3}$
$u := (u_1, \ldots, u_\ell) \in_R \mathcal{G}^\ell$
$\hat{u} := (\hat{u}_0, \hat{u}_1) \in_R \mathcal{G}^2$
$s := (s_1, \ldots, s_\ell) \in_R \mathcal{G}^\ell$
$\hat{s} := (\hat{s}_0, \hat{s}_1) \in_R \mathcal{G}^2$
$x^{\mathsf{U}} := (x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}) := (u_1 \star x_{b_1}, \ldots, u_\ell \star x_{b_\ell})$          $/\!/\mathbf{H_0}$-$\mathbf{H_3}$
$\hat{x}^{\mathsf{U}} := (\hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}) := (\hat{u}_0 \star x_0, \hat{u}_1 \star x_1)$          $/\!/\mathbf{H_0}$-$\mathbf{H_3}$
$x^{\mathsf{S}} := (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}) := (s_1 \star x_{b_1}, \ldots, s_\ell \star x_{b_\ell})$          $/\!/\mathbf{H_0}$-$\mathbf{H_3}$
$\hat{x}^{\mathsf{S}} := (\hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}) := (\hat{s}_0 \star x_0, \hat{s}_1 \star x_1)$          $/\!/\mathbf{H_0}$-$\mathbf{H_3}$
$\mathbf{for}\ i \in [\ell]:$                                                     $/\!/\mathbf{H_0}$-$\mathbf{H_3}$
    $\sigma_i := (\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}) := (u_i \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \star x_i^{\mathsf{S}}, u_i \star \hat{x}_{b_i}^{\mathsf{S}})$ $/\!/\mathbf{H_0}$-$\mathbf{H_3}$
$\sigma := (\sigma_1, \ldots, \sigma_\ell)$                                        $/\!/\mathbf{H_0}$-$\mathbf{H_3}$
$x^{\mathsf{U}} := (x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}) := (u_1 \star \tilde{x}, \ldots, u_\ell \star \tilde{x})$              $/\!/\mathbf{H_4}$
$\hat{x}^{\mathsf{U}} := (\hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}) := (\hat{u}_0 \star \tilde{x}, \hat{u}_1 \star \tilde{x})$              $/\!/\mathbf{H_4}$
$x^{\mathsf{S}} := (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}) := (s_1 \star \tilde{x}, \ldots, s_\ell \star \tilde{x})$              $/\!/\mathbf{H_4}$
$\hat{x}^{\mathsf{S}} := (\hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}) := (\hat{s}_0 \star \tilde{x}, \hat{s}_1 \star \tilde{x})$              $/\!/\mathbf{H_4}$
$\mathbf{if}\ \exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$    $/\!/\mathbf{H_1}$-$\mathbf{H_4}$
    $\mathbf{bad}_{coll} := \mathbf{true}$                                        $/\!/\mathbf{H_1}$-$\mathbf{H_4}$
    $\mathbf{return}\ \perp$                                                      $/\!/\mathbf{H_1}$-$\mathbf{H_4}$
$SK := \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma)$            $/\!/\mathbf{H_0}$-$\mathbf{H_2}$
$SK \in_R \mathcal{SK}$                                                           $/\!/\mathbf{H_3}$-$\mathbf{H_4}$
$\pi_{\mathsf{U}}^{t_0} := ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), SK, \mathbf{true})$
$\pi_{\mathsf{S}}^{t_1} := (s, (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), SK, \mathbf{true})$
$(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) := (\mathbf{true}, \mathbf{true})$                 $/\!/\mathbf{H_2}$-$\mathbf{H_4}$
$\mathbf{return}\ (\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

$\mathsf{REVEAL}(\mathsf{P}, t)$
$\mathbf{if}\ \pi_{\mathsf{P}}^t.acc \neq \mathbf{true}\ \mathbf{or}\ \pi_{\mathsf{P}}^t.test = \mathbf{true}$
    $\mathbf{return}\ \perp$
$\mathbf{if}\ \exists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
  $\mathbf{and}\ \pi_{\mathsf{P}'}^{t'}.test = \mathbf{true}$
    $\mathbf{return}\ \perp$
$\mathbf{if}\ \forall (\mathsf{P}', t')$ s.t. $\pi_{\mathsf{P}'}^{t'}.\mathsf{tr} = \pi_{\mathsf{P}}^t.\mathsf{tr}$                     $/\!/\mathbf{H_2}$-$\mathbf{H_4}$
    $\pi_{\mathsf{P}'}^{t'}.\mathsf{fr} := \mathbf{false}$                         $/\!/\mathbf{H_2}$-$\mathbf{H_4}$
$\mathbf{return}\ \pi_{\mathsf{P}}^t.SK$

$\mathsf{TEST}(\mathsf{P}, t)$
$\mathbf{if}\ \mathsf{Fresh}(\pi_{\mathsf{P}}^t) = \mathbf{false}\ \mathbf{return}\ \perp$                    $/\!/\mathbf{H_0}$-$\mathbf{H_1}$
$\mathbf{if}\ \pi_{\mathsf{P}}^t.\mathsf{fr} := \mathbf{false}\ \mathbf{return}\ \perp$                     $/\!/\mathbf{H_2}$-$\mathbf{H_4}$
$SK_0^* := \mathsf{REVEAL}(\mathsf{P}, t)$
$\mathbf{if}\ SK_0^* = \perp\ \mathbf{return}\ \perp$
$SK_1^* \in_R \mathcal{SK}$
$\pi_{\mathsf{P}}^t.test := \mathbf{true}$
$\mathbf{return}\ K_\beta^*$
```

Fig. 3. Experiments $\mathbf{H_0}$–$\mathbf{H_4}$ for the proof of Theorem 1.

```
$\mathsf{SEND}(\mathsf{P}, t, msg)$
if $msg =$ start:
    if $\pi_{\mathsf{P}}^t \neq \perp$ return $\perp$
    $(b_1, \ldots, b_\ell) := \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}$
    $p := (p_1, \ldots, p_\ell) \in_R \mathcal{G}^\ell$
    $\hat{p} := (\hat{p}_0, \hat{p}_1) \in_R \mathcal{G}^2$
    $x^{\mathsf{P}} := (x_1^{\mathsf{P}}, \ldots, x_\ell^{\mathsf{P}}) := (p_1 \star x_{b_1}, \ldots, p_\ell \star x_{b_\ell})$
    $\hat{x}^{\mathsf{P}} := (\hat{x}_0^{\mathsf{P}}, \hat{x}_1^{\mathsf{P}}) := (\hat{p}_0 \star x_0, \hat{p}_1 \star x_1)$
    $\pi_{\mathsf{P}}^t := ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, \perp, \perp), \perp, \perp)$
    $\pi_{\mathsf{P}}^t.\mathsf{fr} := \perp$                                           //$\mathbf{H_2}$-$\mathbf{H_4}$
    return $(\mathsf{P}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}})$
else if $msg = (\bar{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$:
    if $\pi_{\mathsf{P}}^t \neq ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, \perp, \perp), \perp, \perp)$
        return $\perp$
    if $\exists \mathsf{P}' \in \mathcal{U}, t'$ s.t. $\pi_{\mathsf{P}'}^{t'}.\mathsf{tr} = (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$        //$\mathbf{H_1}$-$\mathbf{H_4}$
      $\mathbf{bad}_{coll} :=$ true                                            //$\mathbf{H_1}$-$\mathbf{H_4}$
        return $\perp$                                                   //$\mathbf{H_1}$-$\mathbf{H_4}$
    if $\exists t'$ s.t. $\pi_{\bar{\mathsf{P}}}^{t'}.\mathsf{tr} = (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$
   and $\pi_{\bar{\mathsf{P}}}^{t'}.\mathsf{fr} =$ true                                        //$\mathbf{H_2}$-$\mathbf{H_4}$
      $\pi_{\mathsf{P}}^t.\mathsf{fr} =$ true                                           //$\mathbf{H_2}$-$\mathbf{H_4}$
    else if $(\mathsf{P}, \bar{\mathsf{P}}) \notin L_{\mathsf{COR}}$                                 //$\mathbf{H_2}$-$\mathbf{H_4}$
        $\pi_{\mathsf{P}}^t.\mathsf{fr} :=$ true                                         //$\mathbf{H_2}$-$\mathbf{H_4}$
    else                                                            //$\mathbf{H_2}$-$\mathbf{H_4}$
        $\pi_{\mathsf{P}}^t.\mathsf{fr} :=$ false                                        //$\mathbf{H_2}$-$\mathbf{H_4}$
      $(b_1, \ldots, b_\ell) := \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}$
        if $\mathsf{P} = \mathsf{U}$:
            for $i \in [\ell]$
                $\sigma_i := (\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}) := (u_i \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \star x_i^{\mathsf{S}}, u_i \star \hat{x}_{b_i}^{\mathsf{S}})$
        else if $\mathsf{P} = \mathsf{S}$:
            for $i \in [\ell]$
                $\sigma_i := (\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}) := (s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_{b_i}^{\mathsf{U}}, \hat{s}_{b_i} \star x_i^{\mathsf{U}})$
        $\sigma := (\sigma_1, \ldots, \sigma_\ell)$
        $SK := \mathsf{H}(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}, \sigma)$
    $\pi_{\mathsf{P}}^t := ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}), SK, \mathbf{true})$
    return true

$\mathsf{CORRUPT}(\mathsf{U}, \mathsf{S})$
if $(\mathsf{U}, \mathsf{S}) \in L_{\mathsf{COR}}$ return $\perp$
for $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$ :
    if $\exists t$ s.t. $\pi_{\mathsf{P}}^t.test =$ true
   and $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
        return $\perp$
    $\forall \pi_{\mathsf{P}}^t$ : if $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$           //$\mathbf{H_2}$-$\mathbf{H_4}$
        $\pi_{\mathsf{P}}^t.\mathsf{fr} =$ false                                        //$\mathbf{H_2}$-$\mathbf{H_4}$
$L_{\mathsf{COR}} := L_{\mathsf{COR}} \cup \{(\mathsf{U}, \mathsf{S})\}$
return $\mathsf{pw}_{\mathsf{US}}$

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma)$
if $L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma] = SK \neq \perp$
    return $SK$
$L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma] \in_R \mathcal{SK}$
return $L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma]$
```

Fig. 4. Experiments $\mathbf{H_0}$–$\mathbf{H_4}$ for the proof of Theorem 1.

We mark the variable as follows: All instances generated by an EXECUTE query are marked as fresh (line 34). In the SEND query, we mark the instance as fresh if the password was not corrupted yet (line 69) and there exists a fresh partner (line 66), otherwise, we mark the instance as not fresh (line 71). If $\mathcal{A}$ issues a CORRUPT query after the instance is generated, the freshness variable is updated (line 88). Also, if the session key of an instance is revealed, this instance and its partner instance are marked as not fresh (line 41). In the TEST query, this freshness variable is checked (line 44). Note that these are conceptual changes, and thus $Adv(\mathcal{A}, \mathbf{H_2}) = Adv(\mathcal{A}, \mathbf{H_1})$.

Hybrid Experiment $\mathbf{H_3}$: In the experiment H_3, we change the way of computation of SK for instances queried to EXECUTE. Instead of computing $SK \leftarrow \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma)$, it is changed as $SK \in_R \mathcal{SK}$. We construct adversary $\mathcal{B}_1$ against GA-StCDH in Fig. 5 from $\mathcal{A}$ in $\mathbf{H_2}$ or $\mathbf{H_3}$.

Initially, $\mathcal{B}_1$ takes $(x, y) = (g\star\tilde{x}, h\star\tilde{x})$ as input and can access $\mathsf{GA\text{-}DDH}(x, \cdot, \cdot)$. First, $\mathcal{B}_1$ generates CRS as in $\mathbf{H_2}$, and simulates this experiment using $\mathcal{A}$. For querying to EXECUTE, $\mathcal{B}_1$ simulates it as follows:

For $i \in [\ell]$, $\mathcal{B}_1$ chooses ephemeral secret values $u_i, \hat{u}_0, \hat{u}_1$ and $s_i, \hat{s}_0, \hat{s}_1$ for each instance. Also, $\mathcal{B}_1$ uses x for U's message generation and y for S's message generation instead of using x_{b_ℓ}. Therefore, the messages in the EXECUTE query are generated independently of the password bits b_i (lines 30 to 33). Note that this message can be rewritten as follows:

$$x_i^{\mathsf{U}} = u_i \star x = (u_i \cdot g) \star \tilde{x} = (u_i \cdot g \cdot g_{b_i} \cdot g_{b_i}^{-1}) \star \tilde{x} = (u_i \cdot g \cdot g_{b_i}^{-1}) \star x_{b_i},$$

where the value $u_i' = u_i \cdot g \cdot g_{b_i}^{-1}$ in brackets is the secret value used for the DH computation in the actual protocol. For the other messages, the values $\hat{u}_0' = \hat{u}_0 \cdot g \cdot g_{b_i}^{-1}$, $\hat{u}_1' = \hat{u}_1 \cdot g \cdot g_{b_i}^{-1}$, $s_i' = s_i \cdot h \cdot g_{b_i}^{-1}$, $\hat{s}_0' = \hat{s}_0 \cdot h \cdot g_{b_i}^{-1}$ and $\hat{s}_1' = \hat{s}_1 \cdot h \cdot g_{b_i}^{-1}$ are the secret value for the DH computation. This means that shared value $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3})$ is implicitly computed as follows:

$$\begin{aligned}
\sigma_{i,1} &= (u_i' \cdot s_i') \star x_{b_i} = u_i \cdot g \cdot s_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x}, \\
\sigma_{i,2} &= (\hat{u}_{b_i}' \cdot s_i') \star x_{b_i} = \hat{u}_{b_i} \cdot g \cdot s_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x}, \\
\sigma_{i,3} &= (u_i' \cdot \hat{s}_{b_i}') \star x_{b_i} = u_i \cdot g \cdot \hat{s}_{b_i} \cdot h \cdot g_{b_i}^{-1} \star \tilde{x}.
\end{aligned}$$

Before choosing a session key at random, we check whether a value $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma)$ matching the true session key is queried to the random oracle H (line 40 to 50). We continue to check in L_{H} whether even one entry in σ is computed correctly with $\mathsf{pw}_{\mathsf{US}}$. We can simulate as follows using the DDH oracle:

$$\begin{aligned}
\mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \sigma_{i,1} &\Leftrightarrow \mathsf{GA\text{-}CDH}(x, x_i^{\mathsf{S}}) = (u_i^{-1} \cdot g_{b_i}) \star \sigma_{i,1}, \\
\mathsf{GA\text{-}CDH}_{x_{b_i}}(\hat{x}_{b_i}^{\mathsf{U}}, x_i^{\mathsf{S}}) = \sigma_{i,2} &\Leftrightarrow \mathsf{GA\text{-}CDH}(x, x_i^{\mathsf{S}}) = (\hat{u}_{b_i}^{-1} \cdot g_{b_i}) \star \sigma_{i,2}, \\
\mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, \hat{x}_{b_i}^{\mathsf{S}}) = \sigma_{i,3} &\Leftrightarrow \mathsf{GA\text{-}CDH}(x, \hat{x}_{b_i}^{\mathsf{S}}) = (u_i^{-1} \cdot g_{b_i}) \star \sigma_{i,3},
\end{aligned}$$

which allows us to use the restricted decision oracle $\mathsf{GA\text{-}DDH}(x, \cdot, \cdot)$. If even one correct shared value exists in σ, $\mathcal{B}_1$ aborts the experiment and outputs $(g \cdot h) \star \tilde{x}$

```
$\mathcal{B}_1^{\mathsf{GA\text{-}DDH}(x,\cdot,\cdot)}(x,y)$
00 $(g_0,g_1) \in_R \mathcal{G}^2$
01 $(x_0,x_1) := (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(L_{\mathsf{H}}, L_{\mathsf{COR}}, L_e) := (\emptyset,\emptyset,\emptyset)$
03 $\mathbf{bad}_{coll} := \mathbf{false}$
04 $\beta \in_R \{0,1\}$
05 for $(\mathsf{U},\mathsf{S}) \in \mathcal{U} \times \mathcal{S}$ :
06    $\mathsf{pw}_{\mathsf{US}} \in_R \mathcal{PW}$
07 $\beta' \leftarrow \mathcal{A}^{\mathcal{O}}(x_0,x_1)$
08 Stop.

$\mathsf{H}(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},\sigma)$
09 if $\exists(u,\hat{u},s,\hat{s})$
   s.t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},u,\hat{u},s,\hat{s}) \in L_e$
10    $(b_1,\ldots,b_\ell) := \mathsf{pw}_{\mathsf{US}}$
11    for $i \in [\ell]$ :
12       $(\sigma_{i,1},\sigma_{i,2},\sigma_{i,3}) := \sigma_i$
13       if $\mathsf{GA\text{-}DDH}(x, x_i^{\mathsf{S}}, (u_i^{-1} \cdot g_{b_i}) \star \sigma_{i,1}) = 1$
14          Stop with $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star \sigma_{i,1}$
15       if $\mathsf{GA\text{-}DDH}(x, x_i^{\mathsf{S}}, (\hat{u}_{b_i}^{-1} \cdot g_{b_i}) \star \sigma_{i,2}) = 1$
16          Stop with $(\hat{u}_{b_i}^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star \sigma_{i,2}$
17       if $\mathsf{GA\text{-}DDH}(x, \hat{x}_{b_i}^{\mathsf{S}}, (u_i^{-1} \cdot g_{b_i}) \star \sigma_{i,3}) = 1$
18          Stop with $(u_i^{-1} \cdot \hat{s}_{b_i}^{-1} \cdot g_{b_i}) \star \sigma_{i,3}$
19 if $L_{\mathsf{H}}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},\sigma] = SK \neq \perp$
20    return $SK$
21 $L_{\mathsf{H}}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},\sigma] \in_R \mathcal{SK}$
22 return $L_{\mathsf{H}}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},\sigma]$
```

```
$\mathsf{EXECUTE}(\mathsf{U},t_0,\mathsf{S},t_1)$
23 if $\pi_{\mathsf{U}}^{t_0} \neq \perp$ or $\pi_{\mathsf{S}}^{t_1} \neq \perp$
24    return $\perp$
25 $(b_1,\ldots,b_\ell) := \mathsf{pw}_{\mathsf{US}}$
26 $u := (u_1,\ldots,u_\ell) \in_R \mathcal{G}^\ell$
27 $\hat{u} := (\hat{u}_0,\hat{u}_1) \in_R \mathcal{G}^2$
28 $s := (s_1,\ldots,s_\ell) \in_R \mathcal{G}^\ell$
29 $\hat{s} := (\hat{s}_0,\hat{s}_1) \in_R \mathcal{G}^2$
30 $x^{\mathsf{U}} := (x_1^{\mathsf{U}},\ldots,x_\ell^{\mathsf{U}}) := (u_1 \star x,\ldots,u_\ell \star x)$
31 $\hat{x}^{\mathsf{U}} := (\hat{x}_0^{\mathsf{U}},\hat{x}_1^{\mathsf{U}}) := (\hat{u}_0 \star x, \hat{u}_1 \star x)$
32 $x^{\mathsf{S}} := (x_1^{\mathsf{S}},\ldots,x_\ell^{\mathsf{S}}) := (s_1 \star y,\ldots,s_\ell \star y)$
33 $\hat{x}^{\mathsf{S}} := (\hat{x}_0^{\mathsf{S}},\hat{x}_1^{\mathsf{S}}) := (\hat{s}_0 \star y, \hat{s}_1 \star y)$
34 if $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}})$
35    $\mathbf{bad}_{coll} := \mathbf{true}$
36    return $\perp$
37 $\forall\sigma$ s.t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},\sigma) \in L_{\mathsf{H}}$
38    for $i \in [\ell]$ :
39       $(\sigma_{i,1},\sigma_{i,2},\sigma_{i,3}) := \sigma_i$
40       if $\mathsf{GA\text{-}DDH}(x, x_i^{\mathsf{S}}, (u_i^{-1} \cdot g_{b_i}) \star \sigma_{i,1}) = 1$
41          Stop with $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star \sigma_{i,1}$
42       if $\mathsf{GA\text{-}DDH}(x, x_i^{\mathsf{S}}, (\hat{u}_{b_i}^{-1} \cdot g_{b_i}) \star \sigma_{i,2}) = 1$
43          Stop with $(\hat{u}_{b_i}^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star \sigma_{i,2}$
44       if $\mathsf{GA\text{-}DDH}(x, \hat{x}_{b_i}^{\mathsf{S}}, (u_i^{-1} \cdot g_{b_i}) \star \sigma_{i,3}) = 1$
45          Stop with $(u_i^{-1} \cdot \hat{s}_{b_i}^{-1} \cdot g_{b_i}) \star \sigma_{i,3}$
46 $L_e := L_e \cup \{\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},u,\hat{u},s\}$
47 $SK \in_R \mathcal{SK}$
48 $\pi_{\mathsf{U}}^{t_0} := ((u,\hat{u}),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}}),\hat{x}^{\mathsf{S}},SK,\mathbf{true})$
49 $\pi_{\mathsf{S}}^{t_1} := (s,(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}),SK,\mathbf{true})$
50 $(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) := (\mathbf{true},\mathbf{true})$
51 return $(\mathsf{U},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},\mathsf{S},x^{\mathsf{S}},\hat{x}^{\mathsf{S}})$
```

Fig. 5. Adversary $\mathcal{B}_1$ against GA-StCDH for the proof of Theorem 1. Oracles SEND,REVEAL,CORRUPT and TEST are the same as in $\mathbf{H_3}$.

which is respectively given by $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star \sigma_{i,1}$, $(\hat{u}_{b_i}^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star \sigma_{i,2}$ or $(u_i^{-1} \cdot \hat{s}_{b_i}^{-1} \cdot g_{b_i}) \star \sigma_{i,3}$.

Otherwise, we store the ephemeral secret values $u_i, \hat{u}_0, \hat{u}_1$ and $s_i, \hat{s}_0, \hat{s}_1$ in L_e, which is to identify relevant queries to H, together with the trace and the password (line 46) and choose a session key uniformly at random. Especially, if the trace and $\mathsf{pw}_{\mathsf{US}}$ are queried with new σ to H, we can re-evaluate in the same way as described above using the DDH oracle by retrieving the secret values from L_e (lines 13 to 18). If the oracle returns 1 for any σ_i, $\mathcal{B}_1$ aborts the experiment and outputs $(g \cdot h) \star \tilde{x}$ which is respectively given by $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star \sigma_{i,1}$, $(\hat{u}_{b_i}^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star \sigma_{i,2}$ or $(u_i^{-1} \cdot \hat{s}_{b_i}^{-1} \cdot g_{b_i}) \star \sigma_{i,3}$.

Thus, since the advantage of $\mathcal{B}_1$ is negligible due to the GA-StCDH assumption, $|Adv(\mathcal{A}, \mathbf{H_3}) - Adv(\mathcal{A}, \mathbf{H_2})| \leq \Pr[\mathbf{bad}_{coll}] \leq negl$.

```
Experiment $\mathbf{H_5}$
$(g_0, g_1) \in_R \mathcal{G}^2$
$(x_0, x_1) := (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
$(L_{\mathsf{H}}, L_{\mathsf{COR}}, L_s) := (\emptyset, \emptyset, \emptyset)$
$\mathbf{bad}_{coll} := \mathbf{false}$
$\beta \in_R \{0, 1\}$
for $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$ :
    $\mathsf{pw}_{\mathsf{US}} \in_R \mathcal{PW}$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}}(x_0, x_1)$
return $1$ or $\perp$

$\mathsf{EXECUTE}(\mathsf{U}, t_0, \mathsf{S}, t_1)$
if $\pi_{\mathsf{U}}^{t_0} \neq \perp$ or $\pi_{\mathsf{S}}^{t_1} \neq \perp$ : return $\perp$
$u := (u_1, \ldots, u_\ell) \in_R \mathcal{G}^\ell$
$\hat{u} := (\hat{u}_0, \hat{u}_1) \in_R \mathcal{G}^2$
$s := (s_1, \ldots, s_\ell) \in_R \mathcal{G}^\ell$
$\hat{s} := (\hat{s}_0, \hat{s}_1) \in_R \mathcal{G}^2$
$x^{\mathsf{U}} := (x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}) := (u_1 \star \tilde{x}, \ldots, u_\ell \star \tilde{x})$
$\hat{x}^{\mathsf{U}} := (\hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}) := (\hat{u}_0 \star \tilde{x}, \hat{u}_1 \star \tilde{x})$
$x^{\mathsf{S}} := (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}) := (s_1 \star \tilde{x}, \ldots, s_\ell \star \tilde{x})$
$\hat{x}^{\mathsf{S}} := (\hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}}) := (\hat{s}_0 \star \tilde{x}, \hat{s}_1 \star \tilde{x})$
if $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}})$
    return $\perp$
$SK \in_R \mathcal{SK}$
$\pi_{\mathsf{U}}^{t_0} := ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}), \hat{x}^{\mathsf{S}}, SK, \mathbf{true})$
$\pi_{\mathsf{S}}^{t_1} := (s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), SK, \mathbf{true})$
$(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) := (\mathbf{true}, \mathbf{true})$
return $(\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma)$
if $L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma] = SK \neq \perp$
    return $SK$
$(b_1, \ldots, b_\ell) := \mathsf{pw}$
if $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}) \in L_s$ and $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$
    if $L_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] = (\mathsf{U}, (u, \hat{u}), SK)$
        for $i \in [\ell]$:
            $\sigma_i' := (u_i \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \star x_i^{\mathsf{S}}, u_i \star \hat{x}_{b_i}^{\mathsf{S}})$
        $\sigma' := (\sigma_1', \ldots, \sigma_\ell')$
    if $L_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] = (\mathsf{S}, (s, \hat{s}), SK)$
        for $i \in [\ell]$:
            $\sigma_i' := (s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_{b_i}^{\mathsf{U}}, \hat{s}_{b_i} \star x_i^{\mathsf{U}})$
        $\sigma' := (\sigma_1', \ldots, \sigma_\ell')$
    if $\sigma = \sigma'$
        if $(\mathsf{U}, \mathsf{S}) \in L_{\mathsf{COR}}$ : return $SK$
        if $(\mathsf{U}, \mathsf{S}) \notin L_{\mathsf{COR}}$ : $\mathbf{bad} := \mathbf{true}$
$L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma] \in_R \mathcal{SK}$
return $L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma]$
```

Fig. 6. Experiment $\mathbf{H_5}$ for the proof of Theorem 1. REVEAL, TEST and CORRUPT are the same as in Fig. 3 and 4.

```
$\underline{\mathsf{SEND}(\mathsf{P}, t, msg)}$
if $msg =$ start:
   if $\pi^t_{\mathsf{P}} \neq \perp$ return $\perp$
   $(b_1, \ldots, b_\ell) := \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}$
   $p := (p_1, \ldots, p_\ell) \in_R \mathcal{G}^\ell$
   $\hat{p} := (\hat{p}_0, \hat{p}_1) \in_R \mathcal{G}^2$
   $x^{\mathsf{P}} := (x^{\mathsf{P}}_1, \ldots, x^{\mathsf{P}}_\ell) := (p_1 \star x_{b_1}, \ldots, p_\ell \star x_{b_\ell})$
   $\hat{x}^{\mathsf{P}} := (\hat{x}^{\mathsf{P}}_0, \hat{x}^{\mathsf{P}}_1) := (\hat{p}_0 \star x_0, \hat{p}_1 \star x_1)$
   $\pi^t_{\mathsf{P}} := ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, \perp, \perp), \perp, \perp)$
   $\pi^t_{\mathsf{P}}.\mathsf{fr} := \perp$
   return $(\mathsf{P}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}})$
else if $msg = (\bar{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$:
   if $\pi^t_{\mathsf{P}} \neq ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, \perp, \perp), \perp, \perp)$
      return $\perp$
   if $\exists \mathsf{P}' \in \mathcal{U}, t'$ s.t. $\pi^{t'}_{\mathsf{P}'}.\mathsf{tr} = (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$
      return $\perp$
   if $\exists t'$ s.t. $\pi^{t'}_{\bar{\mathsf{P}}}.\mathsf{tr} = (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$
  and $\pi^{t'}_{\bar{\mathsf{P}}}.\mathsf{fr} =$ true
      $\pi^t_{\mathsf{P}}.\mathsf{fr} =$ true
      $(\bar{\mathsf{P}}, (\bar{p}, \hat{\bar{p}}), SK) := L_s[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}]$
      if $\exists \sigma$ s.t. $(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}, \sigma) \in L_{\mathsf{H}}$
     and $(\bar{\mathsf{P}} = \mathsf{U}$ and $\sigma_i := (u_i \star x^{\mathsf{S}}_i, \hat{u}_{b_i} \star x^{\mathsf{S}}_i, u_i \star \hat{x}^{\mathsf{S}}_{b_i})\ \forall i \in [\ell])$
     $\cup\ (\bar{\mathsf{P}} = \mathsf{S}$ and $\sigma_i := (s_i \star x^{\mathsf{U}}_i, s_i \star \hat{x}^{\mathsf{U}}_{b_i}, \hat{s}_{b_i} \star x^{\mathsf{U}}_i)\ \forall i \in [\ell])$
         bad := true
   else if $(\mathsf{P}, \bar{\mathsf{P}}) \notin L_{\mathsf{COR}}$
      $\pi^t_{\mathsf{P}}.\mathsf{fr} :=$ true
      if $\exists \sigma$ s.t. $(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}, \sigma) \in L_{\mathsf{H}}$
     and $(\bar{\mathsf{P}} = \mathsf{U}$ and $\sigma_i := (u_i \star x^{\mathsf{S}}_i, \hat{u}_{b_i} \star x^{\mathsf{S}}_i, u_i \star \hat{x}^{\mathsf{S}}_{b_i})\ \forall i \in [\ell])$
     $\cup\ (\bar{\mathsf{P}} = \mathsf{S}$ and $\sigma_i := (s_i \star x^{\mathsf{U}}_i, s_i \star \hat{x}^{\mathsf{U}}_{b_i}, \hat{s}_{b_i} \star x^{\mathsf{U}}_i)\ \forall i \in [\ell])$
         bad := true
      $SK \in_R \mathcal{SK}$
      $L_s[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}] := (\mathsf{P}, (p, \hat{p}), SK)$
   else
      $\pi^t_{\mathsf{P}}.\mathsf{fr} :=$ false
      $(b_1, \ldots, b_\ell) := \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}$
      if $\mathsf{P} = \mathsf{U}$:
         for $i \in [\ell]$
            $\sigma_i := (\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}) := (u_i \star x^{\mathsf{S}}_i, \hat{u}_{b_i} \star x^{\mathsf{S}}_i, u_i \star \hat{x}^{\mathsf{S}}_{b_i})$
      else if $\mathsf{P} = \mathsf{S}$:
         for $i \in [\ell]$
            $\sigma_i := (\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}) := (s_i \star x^{\mathsf{U}}_i, s_i \star \hat{x}^{\mathsf{U}}_{b_i}, \hat{s}_{b_i} \star x^{\mathsf{U}}_i)$
      $\sigma := (\sigma_1, \ldots, \sigma_\ell)$
      $SK := \mathsf{H}(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}, \sigma)$
   $\pi^t_{\mathsf{P}} := ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}), SK,$ true$)$
   return true
```

Fig. 7. Experiment $\mathbf{H_5}$ for the proof of Theorem 1. REVEAL, TEST and CORRUPT are the same as in Fig. 3 and 4.

Hybrid Experiment $\mathbf{H_4}$: In the experiment H_4, we remove the password from $\mathsf{EXECUTE}$ query. Therefore, we use the origin $\tilde{x}$ instead of x_{b_i} for gererating messages $x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}$ and $\hat{x}^{\mathsf{S}}$. These values are the same distribution as in $\mathbf{H_3}$, and the secret values are not used in the derivation of the session key in $\mathbf{H_3}$. Thus, $Adv(\mathcal{A}, \mathbf{H_4}) = Adv(\mathcal{A}, \mathbf{H_3})$.

Hybrid Experiment $\mathbf{H_5}$: In this experiment in Fig. 6 and 7, we change the way of computation of SK for all fresh instances queried to SEND (line 66). Instead of computing $SK \leftarrow \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma)$, it is changed as $SK \in_R \mathcal{SK}$. Here, we add an additional independent random oracle L_s which maps the trace and SK to the secret values of U or S (line 67) to keep partner instances consistent. Specifically, if $\mathcal{A}$ queries SEND for an instance of P and there exists a fresh partner instance, we retrieve the corresponding session key SK from L_s and assign it to the instance (line 59). For all non-fresh instances, we compute the session key using random oracle H according to the protocol (lines 69 to 79).

If an event that a session is fresh and there is an inconsistency between L_{H} and L_s occurs, we set flag **bad**. The event happens in the following cases: (i) For the instance which computes the session key, there exists no partner instance and the password is not corrupted, but the correct password and σ already exist in L_{H} (lines 60 to 65). (ii) the random oracle H is queried on some trace that exists in L_s together with the correct password and σ (lines 28 to 39). At this time, if the password was corrupted and the adversary issues the correct query, we output the session key SK stored in L_s (line 38).

When **bad** does not occur, there is no difference between H_4 and H_5. Thus, $Adv(\mathcal{A}, \mathbf{H_5}) - Adv(\mathcal{A}, \mathbf{H_4}) \leq \Pr[\mathbf{H_5} \Rightarrow \mathbf{bad}]$.

Hybrid Experiment $\mathbf{H_6}$: In this experiment H_6 in Fig. 8 and 9, we remove the password from the SEND query and generate passwords as late as possible. Specifically, we generate the password in the case that the adversary issues a $\mathsf{CORRUPT}$ query (line 21) or it has stopped with output β' (line 07). In the SEND query, we choose ephemeral secret values $p_i, \hat{p}_0, \hat{p}_1$ uniformly at random, and compute messages $x_i^{\mathsf{P}}, \hat{x}_0^{\mathsf{P}}$ and $\hat{x}_1^{\mathsf{P}}$ using the origin element $\tilde{x}$ (lines 46,47) instead of x_{b_i}. Note that this message can be rewritten as follows:

$$x_i^{\mathsf{U}} = u_i \star \tilde{x} = (u_i \cdot g_0^{-1}) \star x_0 = (u_i \cdot g_1^{-1}) \star x_1 = (u_i \cdot g_{b_i}^{-1}) \star x_{b_i},$$

and we use $\tilde{x}$ for generating $\hat{x}_0^{\mathsf{U}}$, $\hat{x}_1^{\mathsf{U}}$, x_i^{S}, $\hat{x}_0^{\mathsf{S}}$ and $\hat{x}_1^{\mathsf{S}}$ in the same way. For all non-fresh instances, we have to compute the actual correct session keys using $\sigma_i = (s_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}}, s_i \cdot g_{b_i}^{-1} \star \hat{x}_{b_i}^{\mathsf{U}}, \hat{s}_{b_i} \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}})$ (line 82) or $\sigma_i = (u_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, u_i \cdot g_{b_i}^{-1} \star x_{b_i}^{\mathsf{S}})$ (line 79). In this case, the password has already been generated in the $\mathsf{CORRUPT}$ query.

```
Experiment $\mathbf{H_6}$
$(g_0, g_1) \in_R \mathcal{G}^2$
$(x_0, x_1) := (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
$(L_{\mathsf{H}}, L_{\mathsf{COR}}, L_s, L_{\mathbf{bad}}) := (\emptyset, \emptyset, \emptyset, \emptyset)$
$(\mathbf{bad}_{guess}, \mathbf{bad}_{\mathsf{pw}}) := (\mathbf{false}, \mathbf{false})$
$\beta \in_R \{0, 1\}$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}}(x_0, x_1)$
for $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \backslash L_{\mathsf{COR}}$ :
    $\mathsf{pw}_{\mathsf{US}} \in_R \mathcal{PW}$
if $\exists \mathsf{pw}, \mathsf{pw}', (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \sigma, \sigma')$
  s.t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma) \in L_{\mathbf{bad}}$
  and $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}', \sigma') \in L_{\mathbf{bad}}$
    $\mathbf{bad}_{\mathsf{pw}} := \mathbf{true}$
else
if $\exists \mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \sigma$
  s.t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, \sigma) \in L_{\mathbf{bad}}$
    $\mathbf{bad}_{guess} := \mathbf{true}$
return $1$ or $\perp$

$\mathsf{CORRUPT}(\mathsf{U}, \mathsf{S})$
if $(\mathsf{U}, \mathsf{S}) \in L_{\mathsf{COR}}$ return $\perp$
for $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$
    if $\exists t$ s.t. $\pi^t_{\mathsf{P}}.test = \mathbf{true}$
    and $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi^t_{\mathsf{P}}, \pi^{t'}_{\mathsf{P}'}) = 1$
        return $\perp$
    $\forall \pi^t_{\mathsf{P}}$ : if $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi^t_{\mathsf{P}}, \pi^{t'}_{\mathsf{P}'}) = 1$
        $\pi^t_{\mathsf{P}}.\mathsf{fr} = \mathbf{false}$
$L_{\mathsf{COR}} := L_{\mathsf{COR}} \cup \{(\mathsf{U}, \mathsf{S})\}$
$\mathsf{pw}_{\mathsf{US}} \in_R \mathcal{PW}$
return $\mathsf{pw}_{\mathsf{US}}$

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma)$
if $L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma] = SK \neq \perp$
    return $SK$
if $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}) \in L_s$
    $(b_1, \ldots, b_\ell) := \mathsf{pw}$
    if $L_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] = (\mathsf{U}, (u, \hat{u}), SK)$
        for $i \in [\ell]$:
            $\sigma'_i := (u_i \cdot g^{-1}_{b_i} \star x^{\mathsf{S}}_i, \hat{u}_{b_i} \cdot g^{-1}_{b_i} \star x^{\mathsf{S}}_i, u_i \cdot g^{-1}_{b_i} \star \hat{x}^{\mathsf{S}}_{b_i})$
        $\sigma' := (\sigma'_1, \ldots, \sigma'_\ell)$
    if $L_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] = (\mathsf{S}, (s, \hat{s}), SK)$
        for $i \in [\ell]$:
            $\sigma'_i := (s_i \cdot g^{-1}_{b_i} \star x^{\mathsf{U}}_i, s_i \cdot g^{-1}_{b_i} \star \hat{x}^{\mathsf{U}}_{b_i}, \hat{s}_{b_i} \cdot g^{-1}_{b_i} \star x^{\mathsf{U}}_i)$
        $\sigma' := (\sigma'_1, \ldots, \sigma'_\ell)$
    if $\sigma = \sigma'$
        if $(\mathsf{U}, \mathsf{S}) \in L_{\mathsf{COR}}$ and $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$ : return $SK$
        if $(\mathsf{U}, \mathsf{S}) \notin L_{\mathsf{COR}}$
            $L_{\mathbf{bad}} := L_{\mathbf{bad}} \cup \{\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma\}$
$L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma] \in_R \mathcal{SK}$
return $L_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, \sigma]$
```

Fig. 8. Experiment $\mathbf{H_6}$ for the proof of Theorem 1. REVEAL and TEST are the same as in Fig. 3 and 4. EXECUTE is the same as in Fig. 6 and 7.

```
$\mathsf{SEND}(\mathsf{P}, t, msg)$
if $msg = \textbf{start}$:
    if $\pi_{\mathsf{P}}^t \neq \perp$ return $\perp$
    $(b_1, \ldots, b_\ell) := \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}$
    $p := (p_1, \ldots, p_\ell) \in_R \mathcal{G}^\ell$
    $\hat{p} := (\hat{p}_0, \hat{p}_1) \in_R \mathcal{G}^2$
    $x^{\mathsf{P}} := (x_1^{\mathsf{P}}, \ldots, x_\ell^{\mathsf{P}}) := (p_1 \star \tilde{x}, \ldots, p_\ell \star \tilde{x})$
    $\hat{x}^{\mathsf{P}} := (\hat{x}_0^{\mathsf{P}}, \hat{x}_1^{\mathsf{P}}) := (\hat{p}_0 \star \tilde{x}, \hat{p}_1 \star \tilde{x})$
    $\pi_{\mathsf{P}}^t := ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, \perp, \perp), \perp, \perp)$
    $\pi_{\mathsf{P}}^t.\mathsf{fr} := \perp$
    return $(\mathsf{P}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}})$
else if $msg = (\bar{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$:
    if $\pi_{\mathsf{P}}^t \neq ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, \perp, \perp), \perp, \perp)$
        return $\perp$
    if $\exists \mathsf{P}' \in \mathcal{U}, t'$ s.t. $\pi_{\mathsf{P}'}^{t'}.\mathsf{tr} = (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$
        return $\perp$
    if $\exists t'$ s.t. $\pi_{\bar{\mathsf{P}}}^{t'}.\mathsf{tr} = (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$
      and $\pi_{\bar{\mathsf{P}}}^{t'}.\mathsf{fr} = \textbf{true}$
        $\pi_{\mathsf{P}}^t.\mathsf{fr} = \textbf{true}$
        $(\bar{\mathsf{P}}, (\bar{p}, \hat{\bar{p}}), SK) := L_s[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}]$
    else if $(\mathsf{P}, \bar{\mathsf{P}}) \notin L_{\mathsf{COR}}$
        $\pi_{\mathsf{P}}^t.\mathsf{fr} := \textbf{true}$
        if $\exists \mathsf{pw}, \sigma$ s.t. $(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma) \in L_{\mathsf{H}}$
          and $\mathsf{P} = \mathsf{U}$
            $(b_1, \ldots, b_\ell) := \mathsf{pw}$
            for $i \in [\ell]$ :
                $\sigma'_i := (u_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, u_i \cdot g_{b_i}^{-1} \star \hat{x}_{b_i}^{\mathsf{S}})$
        else if $\exists \mathsf{pw}, \sigma$ s.t. $(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma) \in L_{\mathsf{H}}$
          and $\mathsf{P} = \mathsf{S}$
            $(b_1, \ldots, b_\ell) := \mathsf{pw}$
            for $i \in [\ell]$ :
                $\sigma_i := (s_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}}, s_i \cdot g_{b_i}^{-1} \star \hat{x}_{b_i}^{\mathsf{U}}, \hat{s}_{b_i} \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}})$
        $\sigma' := (\sigma'_1, \ldots, \sigma'_\ell)$
        if $\sigma = \sigma'$
            $L_{\mathbf{bad}} := L_{\mathbf{bad}} \cup \{\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma\}$
        $SK \in_R \mathcal{SK}$
    $L_s[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}] := (\mathsf{P}, (p, \hat{p}), SK)$
    else
        $\pi_{\mathsf{P}}^t.\mathsf{fr} := \textbf{false}$
        $(b_1, \ldots, b_\ell) := \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}$
        if $\mathsf{P} = \mathsf{U}$
            for $i \in [\ell]$ :
                $\sigma_i := (u_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, u_i \cdot g_{b_i}^{-1} \star \hat{x}_{b_i}^{\mathsf{S}})$
        else if $\mathsf{P} = \mathsf{S}$
            for $i \in [\ell]$ :
                $\sigma_i := (s_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}}, s_i \cdot g_{b_i}^{-1} \star \hat{x}_{b_i}^{\mathsf{U}}, \hat{s}_{b_i} \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}})$
        $\sigma := (\sigma_1, \ldots, \sigma_\ell)$
        $SK := \mathsf{H}(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}, \sigma)$
    $\pi_{\mathsf{P}}^t := ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}), SK, \textbf{true})$
    return true
```

Fig. 9. Experiment $\mathbf{H_6}$ for the proof of Theorem 1. REVEAL and TEST are the same as in Fig. 3 and 4. EXECUTE is the same as in Fig. 6 and 7.

We recall that the event **bad** in $\mathbf{H_5}$ occurs in the case that there is an inconsistency in the querying to the random oracle H and the session key of fresh instances. In this experiment, we split the event **bad** into the following two events: (i)$\mathbf{bad}_{\mathsf{pw}}$ which captures the event that there exists more than one correct entry in L_{H} for the same trace of a fresh instance, but the passwords are different. (ii) $\mathbf{bad}_{guess}$ which captures the event that $\mathbf{bad}_{\mathsf{pw}}$ does not occur and there exists a correct entry with the trace of a fresh instance and the correct password in L_{H}, where the password was not corrupted when the query to H was issued.

In order to identify these events, we add a list $L_{\mathbf{bad}}$. For all fresh instances in SEND, we iterate the following for all entries in L_{H}: We check if the given password and σ are correct for the trace by computing the correct values σ' dependent on the password in the same way as for non-fresh instances. If $\sigma = \sigma'$, we add the entry to $L_{\mathbf{bad}}$ (lines 61 to 71). Also, we simulate the random oracle H queried on a trace that appears in L_s in the same way. In this case, $\mathcal{A}$ specifies pw, thus we check whether σ is computed correctly with the pw by using $p_i, \hat{p}_0, \hat{p}_1$ stored in L_s for the instance of P. If σ is correct and the instance is fresh, we add the entry to $L_{\mathbf{bad}}$ (lines 25 to 38). In the case that the pw has been already corrupted, we return the session key stored in L_s. When $\mathcal{A}$ outputs β', we check $L_{\mathbf{bad}}$ whether the event $\mathbf{bad}_{\mathsf{pw}}$ (line 09) or the event $\mathbf{bad}_{guess}$ (line 12) occurred.

Note that when the flag **bad** is set in $\mathbf{H_5}$, the flag $\mathbf{bad}_{guess}$ or $\mathbf{bad}_{\mathsf{pw}}$ is set in $\mathbf{H_6}$, thus $\Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \leq \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] + \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{guess}]$. We evaluate the probability of two events. First, we evaluate the probability of $\mathbf{bad}_{\mathsf{pw}}$. We construct adversary $\mathcal{B}_2$ against $\mathsf{DSim\text{-}GA\text{-}StCDH}$ that simulates $\mathbf{H_6}$ in Fig. 10 and 11.

Initially, $\mathcal{B}_2$ inputs (x_0, x_1, w_0, w_1), where $g_0, g_1, h_0, h_1 \in_R \mathcal{G}$, $x_0 = g_0 \star \tilde{x}$, $x_1 = g_1 \star \tilde{x}$, $w_0 = h_0 \star \tilde{x}$ and $w_1 = h_1 \star \tilde{x}$. $\mathcal{B}_2$ also can access to the DDH oracles $\mathsf{GA\text{-}DDH}_{x_j}(w_k, \cdot, \cdot)$ for $(j,k) \in \{0,1\}^2$. $\mathcal{B}_2$ gives $\mathcal{A}$ (x_0, x_1) as input and simulates SEND queries as follows:

In the case of $\mathsf{P} = \mathsf{U}$, $\mathcal{B}_2$ chooses ephemeral secret values u_i and $\hat{u}_i$ uniformly at random and computes as follows:

$$x_i^{\mathsf{U}} = u_i \star w_0 = (u_i \cdot h_0 \cdot g_0^{-1}) \star x_0 = (u_i \cdot h_0 \cdot g_1^{-1}) \star x_1,$$
$$\hat{x}_0^{\mathsf{U}} = \hat{u}_0 \star w_1 = (\hat{u}_0 \cdot h_1 \cdot g_0^{-1}) \star x_0, \hat{x}_1^{\mathsf{U}} = \hat{u}_1 \star w_1 = (\hat{u}_1 \cdot h_1 \cdot g_1^{-1}) \star x_1.$$

Also, in the case of $\mathsf{P} = \mathsf{S}$, $\mathcal{B}_2$ generates messages $x_i^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}$ and $\hat{x}_1^{\mathsf{S}}$ in the same way.

If the instance of S is fresh, $\mathcal{B}_2$ check if there exists an entry in L_{H} that causes the event **bad**. It iterates over all the pair of (pw, σ) in L_{H} with the trace of the instance. Specifically, it checks whether there is a correct shared value as follows:

$$\sigma_{i,1} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) \Leftrightarrow \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_0, x_i^{\mathsf{U}}) = s_i^{-1} \star \sigma_{i,1},$$
$$\sigma_{i,2} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(\hat{x}_{b_i}^{\mathsf{U}}, x_i^{\mathsf{S}}) \Leftrightarrow \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_0, \hat{x}_{b_i}^{\mathsf{U}}) = s_i^{-1} \star \sigma_{i,2},$$
$$\sigma_{i,3} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, \hat{x}_{b_i}^{\mathsf{S}}) \Leftrightarrow \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_1, x_i^{\mathsf{U}}) = \hat{s}_{b_i}^{-1} \star \sigma_{i,3},$$

where it simulates with the DDH oracles $\mathsf{GA\text{-}DDH}_{x_{b_j}}(z_{i,1}, \cdot, \cdot)$. If $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3})$ are correct, $\mathcal{B}_2$ add the entry to $L_{\mathbf{bad}}$ (lines 61 to 64).

```
$\mathcal{B}_2^{\{\mathsf{GA\text{-}DDH}_{x_j}(w_k,\cdot,\cdot)\}_{j,k\in\{0,1\}}}(x_0, x_1, w_0, w_1)$
$(L_\mathsf{H}, L_\mathsf{COR}, L_s, L_\mathbf{bad}) := (\emptyset, \emptyset, \emptyset, \emptyset)$
$\beta \in_R \{0, 1\}$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}}(x_0, x_1)$
for $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \backslash L_\mathsf{COR}$ :
    $\mathsf{pw}_\mathsf{US} \in_R \mathcal{PW}$
if $\exists \mathsf{pw}, \mathsf{pw}', (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \sigma, \sigma')$
  s.t. $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, \sigma) \in L_\mathbf{bad}$
  and $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}', \sigma') \in L_\mathbf{bad}$
    $(b_1, \ldots, b_\ell) := \mathsf{pw}$
    $(b'_1, \ldots, b'_\ell) := \mathsf{pw}'$
    Find first index $i^*$ such that $b_{i^*} \neq b'_{i^*}$
    W.l.o.g. let $b_{i^*} = 0, b'_{i^*} = 1$
    if $L_s[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}] = (\mathsf{U}, (u, \hat{u}), SK)$
        Stop with $(x^\mathsf{S}_{i^*}, u^{-1}_{i^*} \star \sigma_{i^*,1}, \hat{u}^{-1}_{b_{i^*}} \star \sigma_{i^*,2}, u^{-1}_{i^*} \star \sigma'_{i^*,1}, \hat{u}^{-1}_{b'_{i^*}} \star \sigma'_{i^*,2})$
    if $L_s[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}] = (\mathsf{S}, (s, \hat{s}), SK)$
        Stop with $(x^\mathsf{U}_{i^*}, s^{-1}_{i^*} \star \sigma_{i^*,1}, \hat{s}^{-1}_{b_{i^*}} \star \sigma_{i^*,3}, s^{-1}_{i^*} \star \sigma'_{i^*,1}, \hat{s}^{-1}_{b'_{i^*}} \star \sigma'_{i^*,3})$

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, \sigma)$
if $L_\mathsf{H}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, \sigma] = SK \neq \perp$
    return $SK$
if $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}) \in L_s$
    $(b_1, \ldots, b_\ell) := \mathsf{pw}$
    if $L_s[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}] = (\mathsf{U}, (u, \hat{u}), SK)$
        if $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, x^\mathsf{S}_i, u^{-1}_i \star \sigma_{i,1}) = 1 \ \forall i \in [\ell]$
       and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_1, x^\mathsf{S}_i, \hat{u}^{-1}_{b_i} \star \sigma_{i,2}) = 1 \ \forall i \in [\ell]$
       and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, \hat{x}^\mathsf{S}_{b_i}, u^{-1} \star \sigma_{i,3}) = 1 \ \forall i \in [\ell]$
            if $(\mathsf{U}, \mathsf{S}) \notin L_\mathsf{COR}$
                $L_\mathbf{bad} := L_\mathbf{bad} \cup \{\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, \sigma\}$
            if $(\mathsf{U}, \mathsf{S}) \in L_\mathsf{COR}$ and $\mathsf{pw} = \mathsf{pw}_\mathsf{US}$
                return $SK$
    if $L_s[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}] = (\mathsf{S}, (s, \hat{s}), SK)$
        if $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, x^\mathsf{U}_i, s^{-1}_i \star \sigma_{i,1}) = 1 \ \forall i \in [\ell]$
       and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, \hat{x}^\mathsf{U}_{b_i}, s^{-1}_i \star \sigma_{i,2}) = 1 \ \forall i \in [\ell]$
       and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_1, x^\mathsf{U}_i, \hat{s}^{-1}_{b_i} \star \sigma_{i,3}) = 1 \ \forall i \in [\ell]$
            if $(\mathsf{U}, \mathsf{S}) \notin L_\mathsf{COR}$
                $L_\mathbf{bad} := L_\mathbf{bad} \cup \{\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, \sigma\}$
            if $(\mathsf{U}, \mathsf{S}) \in L_\mathsf{COR}$ and $\mathsf{pw} = \mathsf{pw}_\mathsf{US}$
                return $SK$
if $\exists (u, \hat{u})$ s.t. $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, (u, \hat{u})) \in L_\mathsf{H}$
    $(b_1, \ldots, b_\ell) := \mathsf{pw}$
    if $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, x^\mathsf{S}_i, u^{-1}_i \star \sigma_{i,1}) = 1 \ \forall i \in [\ell]$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_1, x^\mathsf{S}_i, \hat{u}^{-1}_{b_i} \star \sigma_{i,2}) = 1 \ \forall i \in [\ell]$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, \hat{x}^\mathsf{S}_{b_i}, u^{-1}_i \star \sigma_{i,3}) = 1 \ \forall i \in [\ell]$
        return $L_\mathsf{H}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, (u, \hat{u})]$
else if $\exists (s, \hat{s})$
   s.t. $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, (s, \hat{s})) \in L_\mathsf{H}$
    $(b_1, \ldots, b_\ell) := \mathsf{pw}$
   if $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, x^\mathsf{U}_i, s^{-1}_i \star \sigma_{i,1}) = 1 \ \forall i \in [\ell]$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, \hat{x}^\mathsf{U}_{b_i}, s^{-1}_i \star \sigma_{i,2}) = 1 \ \forall i \in [\ell]$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_1, x^\mathsf{U}_i, \hat{s}^{-1}_{b_i} \star \sigma_{i,3}) = 1 \ \forall i \in [\ell]$
        return $L_\mathsf{H}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, (s, \hat{s})]$
$L_\mathsf{H}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, \sigma] \in_R \mathcal{SK}$
return $L_\mathsf{H}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, \sigma]$
```

Fig. 10. Adversary $\mathcal{B}_2$ against $\mathsf{DSim\text{-}GA\text{-}StCDH}$ for the proof of Theorem 1. Oracles $\mathsf{EXECUTE}$,REVEAL,$\mathsf{CORRUPT}$ and TEST are the same as in $\mathbf{H_5}$.

```
$\mathsf{SEND}(\mathsf{P}, t, msg)$
if $msg = \mathbf{start}$:
  if $\pi_{\mathsf{P}}^t \neq \perp$ return $\perp$
  $p := (p_1, \dots, p_\ell) \in_R \mathcal{G}^\ell$
  $\hat{p} := (\hat{p}_0, \hat{p}_1) \in_R \mathcal{G}^2$
  $x^{\mathsf{P}} := (x_1^{\mathsf{P}}, \dots, x_\ell^{\mathsf{P}}) := (p_1 \star w_0, \dots, p_\ell \star w_0)$
  $\hat{x}^{\mathsf{P}} := (\hat{x}_0^{\mathsf{P}}, \hat{x}_1^{\mathsf{P}}) := (\hat{p}_0 \star w_1, \hat{p}_1 \star w_1)$
  $\pi_{\mathsf{P}}^t := ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, \perp, \perp), \perp, \perp)$
  return $(\mathsf{P}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}})$
else if $msg = (\bar{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$:
  if $\pi_{\mathsf{P}}^t \neq ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, \perp, \perp), \perp, \perp)$ : return $\perp$
  if $\exists \mathsf{P}' \in \mathcal{U}, t'$ s.t. $\pi_{\mathsf{P}'}^{t'}.\mathsf{tr} = (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$
    return $\perp$
  if $\exists t'$ s.t. $\pi_{\bar{\mathsf{P}}}^{t'}.\mathsf{tr} = (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}})$ and $\pi_{\bar{\mathsf{P}}}^{t'}.\mathsf{fr} = \mathbf{true}$
    $\pi_{\mathsf{P}}^t.\mathsf{fr} = \mathbf{true}$
    $(\bar{\mathsf{P}}, (\bar{p}, \hat{\bar{p}}), SK) := L_s[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}]$
  else if $(\mathsf{P}, \bar{\mathsf{P}}) \notin L_{\mathsf{COR}}$
    $\pi_{\mathsf{P}}^t.\mathsf{fr} := \mathbf{true}$
    if $\exists \mathsf{pw}, \sigma$ s.t. $(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma) \in L_{\mathsf{H}}$ and $\mathsf{P} = \mathsf{U}$
      $(b_1, \dots, b_\ell) := \mathsf{pw}$
      if $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, x_i^{\mathsf{S}}, u_i^{-1} \star \sigma_{i,1}) = 1\ \forall i \in [\ell]$
     and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_1, x_i^{\mathsf{S}}, \hat{u}_{b_i}^{-1} \star \sigma_{i,2}) = 1\ \forall i \in [\ell]$
     and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, \hat{x}_{b_i}^{\mathsf{S}}, u_i^{-1} \star \sigma_{i,3}) = 1\ \forall i \in [\ell]$
        $L_{\mathbf{bad}} := L_{\mathbf{bad}} \cup \{\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma\}$
    else if $\exists \mathsf{pw}, \sigma$ s.t. $(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma) \in L_{\mathsf{H}}$ and $\mathsf{P} = \mathsf{S}$
      $(b_1, \dots, b_\ell) := \mathsf{pw}$
      if $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, x_i^{\mathsf{U}}, s_i^{-1} \star \sigma_{i,1}) = 1\ \forall i \in [\ell]$
     and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, \hat{x}_{b_i}^{\mathsf{U}}, s_i^{-1} \star \sigma_{i,2}) = 1\ \forall i \in [\ell]$
     and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_1, x_i^{\mathsf{U}}, \hat{s}_{b_i}^{-1} \star \sigma_{i,3}) = 1\ \forall i \in [\ell]$
        $L_{\mathbf{bad}} := L_{\mathbf{bad}} \cup \{\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma\}$
    $SK \in_R \mathcal{SK}$
    $L_s[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}] = (\mathsf{P}, (p, \hat{p}), SK)$
  else
    $\pi_{\mathsf{P}}^t.\mathsf{fr} := \mathbf{false}$
    $(b_1, \dots, b_\ell) := \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}$
    if $\exists \sigma$ s.t. $(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma) \in L_{\mathsf{H}}$ and $\mathsf{P} = \mathsf{U}$
    and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, x_i^{\mathsf{S}}, u_i^{-1} \star \sigma_{i,1}) = 1\ \forall i \in [\ell]$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_1, x_i^{\mathsf{S}}, \hat{u}_{b_i}^{-1} \star \sigma_{i,2}) = 1\ \forall i \in [\ell]$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, \hat{x}_{b_i}^{\mathsf{S}}, u_i^{-1} \star \sigma_{i,3}) = 1\ \forall i \in [\ell]$
      $SK := L_{\mathsf{H}}[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}, \sigma]$
    else if $\exists \sigma$ s.t. $(\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}, \sigma) \in L_{\mathsf{H}}$ and $\mathsf{P} = \mathsf{S}$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, x_i^{\mathsf{U}}, s_i^{-1} \star \sigma_{i,1}) = 1\ \forall i \in [\ell]$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_0, \hat{x}_{b_i}^{\mathsf{U}}, s_i^{-1} \star \sigma_{i,2}) = 1\ \forall i \in [\ell]$
   and $\mathsf{GA\text{-}DDH}_{x_{b_i}}(w_1, x_i^{\mathsf{U}}, \hat{s}_{b_i}^{-1} \star \sigma_{i,3}) = 1\ \forall i \in [\ell]$
      $SK := L_{\mathsf{H}}[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}, \sigma]$
    else
      $SK \in_R \mathcal{SK}$
      $L_{\mathsf{H}}[\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}, \mathsf{pw}_{\mathsf{P}\bar{\mathsf{P}}}, (p, \hat{p})] := SK$
  $\pi_{\mathsf{P}}^t := ((p, \hat{p}), (\mathsf{P}, \bar{\mathsf{P}}, x^{\mathsf{P}}, \hat{x}^{\mathsf{P}}, x^{\bar{\mathsf{P}}}, \hat{x}^{\bar{\mathsf{P}}}), SK, \mathbf{true})$
  return true
```

Fig. 11. Adversary $\mathcal{B}_2$ against $\mathsf{DSim\text{-}GA\text{-}StCDH}$ for the proof of Theorem 1. Oracles $\mathsf{EXECUTE}$,REVEAL,$\mathsf{CORRUPT}$ and TEST are the same as in $\mathbf{H_5}$.

For non-fresh instances, $\mathcal{B}_2$ must compute the correct session key. $\mathcal{B}_2$ checks whether a correct shared value σ exists in L_{H} as described above. If it exists, $\mathcal{B}_2$ assigns SK in L_{H} to the session key (line 72). Otherwise, it chooses a session key at random and add a irregular entry to L_{H}, which contains the elements $(p, \hat{p})$ instead of the shared value σ (line 77), to patch the query to the random oracle H later. In the same way, for the case of the instance of U, it checks σ using the secret group elements u_i, $\hat{u}_0$ and $\hat{u}_1$ (lines 57 to 60) as follows:

$$\sigma_{i,1} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) \Leftrightarrow \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_0, x_i^{\mathsf{S}}) = u_i^{-1} \star \sigma_{i,1},$$
$$\sigma_{i,2} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(\hat{x}_{b_i}^{\mathsf{U}}, x_i^{\mathsf{S}}) \Leftrightarrow \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_1, x_i^{\mathsf{S}}) = \hat{u}_{b_i}^{-1} \star \sigma_{i,2},$$
$$\sigma_{i,3} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, \hat{x}_{b_i}^{\mathsf{S}}) \Leftrightarrow \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_1, \hat{x}_{b_i}^{\mathsf{S}}) = u_i^{-1} \star \sigma_{i,3},$$

In the query to the random oracle H, if the trace is contained in L_{s} which means the corresponding instance was fresh when the SEND query was issued, $\mathcal{B}_2$ checks if σ is valid using the $\mathsf{GA\text{-}DDH}$ oracle as described above (lines 18 and 29). If the shared value σ is valid, it checks whether the instance is fresh, i.e., the password is not corrupted. If this is the case, it adds the query to $L_{\mathbf{bad}}$ (lines 21 and 27). Otherwise, if the password was corrupted and is specified in the query, it returns the session key stored in L_s (lines 23 and 29). After that, it checks whether the queried entry matches the irregular entry added to L_{H} in the previous SEND query for a non-fresh instance, i.e., it must return the same session key as returned previously. Then, for the matching entry, it uses the $\mathsf{GA\text{-}DDH}$ oracle to check depending on whether it is an instance of U or an instance of S (line 32 to 36).

After $\mathcal{A}$ guesses the β', $\mathcal{B}_2$ chooses passwords for users that have not been $\mathsf{CORRUPT}$ queried. Then, it checks whether $\mathbf{bad}_{\mathsf{pw}}$ occurred (lines 05 to 13). If $\mathbf{bad}_{\mathsf{pw}}$ occurs, there must be two entries in $L_{\mathbf{bad}}$ for the same trace and different passwords $\mathsf{pw} \neq \mathsf{pw}'$ along with values σ and σ', and $\mathcal{B}_2$ answers to $\mathsf{DSim\text{-}GA\text{-}StCDH}$ using these entries as follows: First, it compares the bits of the two password bits from the beginning, and let i^* be the first index where the bit values are different, i.e., $b_{i^*} \neq b'_{i^*}$. Then, it assumes that $b_{i^*} = 0$ and $b'_{i^*} = 1$, but it swaps pw, σ and pw', σ' if the assignment is different. Next, if the entries in $L_{\mathbf{bad}}$ are those of an instance of U, $\mathcal{B}_2$ retrieves the ephemeral secret values $u_{i^*}, \hat{u}_{b_{i^*}}$ from L_s and sets $y = x_{i^*}^{\mathsf{S}}$, and outputs y and

$$y_0 = u_{i^*}^{-1} \star \sigma_{i^*,1} = \mathsf{GA\text{-}CDH}_{x_0}(u_{i^*}^{-1} \star x_{i^*}^{\mathsf{U}}, x_{i^*}^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_0}(w_0, x_{i^*}^{\mathsf{S}}),$$
$$y_1 = \hat{u}_{b_{i^*}}^{-1} \star \sigma_{i^*,2} = \mathsf{GA\text{-}CDH}_{x_0}(\hat{u}_{b_{i^*}}^{-1} \star \hat{x}_{b_{i^*}}^{\mathsf{U}}, x_{i^*}^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_0}(w_1, x_{i^*}^{\mathsf{S}}),$$
$$y_2 = u_{i^*}^{-1} \star \sigma'_{i^*,1} = \mathsf{GA\text{-}CDH}_{x_1}(u_{i^*}^{-1} \star x_{i^*}^{\mathsf{U}}, x_{i^*}^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_1}(w_0, x_{i^*}^{\mathsf{S}}),$$
$$y_3 = \hat{u}_{b'_{i^*}}^{-1} \star \sigma'_{i^*,2} = \mathsf{GA\text{-}CDH}_{x_1}(\hat{u}_{b'_{i^*}}^{-1} \star \hat{x}_{b'_{i^*}}^{\mathsf{U}}, x_{i^*}^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_1}(w_1, x_{i^*}^{\mathsf{S}}).$$

Also, if the instance is an instance of S, $\mathcal{B}_2$ retrieves the ephemeral secret values $s_{i^*}, \hat{s}_{b_{i^*}}$ from L_s and sets $y = x_{i^*}^{\mathsf{U}}$, and outputs y and

$$\begin{aligned}
y_0 &= s_{i^*}^{-1} \star \sigma_{i^*,1} = \mathsf{GA\text{-}CDH}_{x_0}(s_{i^*}^{-1} \star x_{i^*}^{\mathsf{S}}, x_{i^*}^{\mathsf{U}}) = \mathsf{GA\text{-}CDH}_{x_0}(w_0, x_{i^*}^{\mathsf{U}}),\\
y_1 &= \hat{s}_{b_{i^*}}^{-1} \star \sigma_{i^*,3} = \mathsf{GA\text{-}CDH}_{x_0}(\hat{s}_{b_{i^*}}^{-1} \star \hat{x}_{b_{i^*}}^{\mathsf{S}}, x_{i^*}^{\mathsf{U}}) = \mathsf{GA\text{-}CDH}_{x_0}(w_1, x_{i^*}^{\mathsf{U}}),\\
y_2 &= s_{i^*}^{-1} \star \sigma'_{i^*,1} = \mathsf{GA\text{-}CDH}_{x_1}(s_{i^*}^{-1} \star x_{i^*}^{\mathsf{S}}, x_{i^*}^{\mathsf{U}}) = \mathsf{GA\text{-}CDH}_{x_1}(w_0, x_{i^*}^{\mathsf{U}}),\\
y_3 &= \hat{s}_{b'_{i^*}}^{-1} \star \sigma'_{i^*,3} = \mathsf{GA\text{-}CDH}_{x_1}(\hat{s}_{b'_{i^*}}^{-1} \star \hat{x}_{b'_{i^*}}^{\mathsf{S}}, x_{i^*}^{\mathsf{U}}) = \mathsf{GA\text{-}CDH}_{x_1}(w_1, x_{i^*}^{\mathsf{U}}).
\end{aligned}$$

This concludes the analysis of $\mathbf{bad}_{\mathsf{pw}}$. Thus, if $\mathbf{bad}_{\mathsf{pw}}$ occurs, $\mathcal{B}_2$ can solve $\mathsf{DSim\text{-}GA\text{-}StCDH}$, i.e., $\Pr[\mathbf{H_6} \Rightarrow \mathbf{bad}_{\mathsf{pw}}] \leq Adv_{\mathsf{EGAT}}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}(\mathcal{B}_2)$.

Second, we evaluate the probability of $\mathbf{bad}_{guess}$. Note that $\mathbf{bad}_{guess}$ occurs only if $\mathbf{bad}_{\mathsf{pw}}$ does not occur. Thus, there is only one entry in $L_{\mathbf{bad}}$ for each instance and the size of $L_{\mathbf{bad}}$ is at most q_s. All instances were generated before the corresponding password was chosen, thus

$$\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{guess}] \leq \frac{q_s}{|\mathcal{PW}|}.$$

In $\mathbf{H_6}$, if none of the bad events occurs, all session keys output by TEST are perfectly randomized. This gives $\mathcal{A}$ no information from the Test query, therefore $Adv(\mathcal{A}, \mathbf{H_6}) = \frac{1}{2}$. □

References

1. Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the J-PAKE password-authenticated key exchange protocol. In: IEEE Symposium on Security and Privacy 2015, pp. 571–587 (2015)
2. Abdalla, M., Eisenhofer, T., Kiltz, E., Kunzweiler, S., Riepel, D.: Password-authenticated key exchange from group actions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022. LNCS, vol. 13508, pp. 699–728. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15979-4_24
3. Abdalla, M., Fouque, P.-A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30580-4_6
4. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30574-3_14
5. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 411–439. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_14
6. Azarderakhsh, R., Jao, D., Koziel, B., LeGrow, J.T., Soukharev, V., Taraskin, O.: How not to create an isogeny-based PAKE. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 2020. LNCS, vol. 12146, pp. 169–186. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57808-4_9

7. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_11
8. Canetti, R., Dachman-Soled, D., Vaikuntanathan, V., Wee, H.: Efficient password authenticated key exchange via oblivious transfer. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 449–466. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_27
9. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_24
10. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version). IACR Cryptology ePrint Archive, Report 2022/975 (2022)
11. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
12. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 524–543. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_33
13. Groce, A., Katz, J.: A new framework for efficient password-based authenticated key exchange. In: ACM CCS 2010, pp. 516–525 (2010)
14. Haase, B., Labrique, B.: AuCPace: efficient verifier-based PAKE protocol tailored for the IIoT. IACR TCHES **2019**, 1–48 (2019)
15. Hao, F., Ryan, P.: J-PAKE: Authenticated key exchange without PKI. IACR Cryptology ePrint Archive, Report 2010/190 (2010)
16. Jablon, D.P.: Strong password-only authenticated key exchange. Comput. Commun. Rev. **26**, 5–26 (1996)
17. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
18. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 293–310. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_18
19. Soukharev, V., Hess, B.: PQDH: A Quantum-Safe Replacement for Diffie-Hellman based on SIDH. IACR Cryptology ePrint Archive, Report 2019/730 (2019)
20. Steven, M., Bellovin, Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: IEEE Computer Society Symposium on Research in Security and Privacy 1992, pp. 72–84 (1992)
21. Taraskin, O., Soukharev, V., Jao, D., LeGrow, J.T.: An Isogeny-Based Password-Authenticated Key Establishment Protocol. IACR Cryptology ePrint Archive, Report 2018/886 (2018)
22. Terada, S., Yoneyama, K.: Password-based authenticated key exchange from standard isogeny assumptions. In: Steinfeld, R., Yuen, T.H. (eds.) ProvSec 2019. LNCS, vol. 11821, pp. 41–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31919-9_3

Multi-key Homomorphic Secret Sharing from LWE Without Multi-key HE

Peiying Xu[1,2,3] and Li-Ping Wang[1,2,3](✉)

[1] State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing, China
{xupeiying,wangliping}@iie.ac.cn

[2] State Key Laboratory of Cryptology, Beijing, China

[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. Homomorphic secret sharing (HSS) allows participants to share their private data between computing servers for the joint computation of a public function without fully homomorphic encryption. It serves as a competitive alternative to somewhat/fully homomorphic encryption (S/FHE) in some scenarios. However, the existing HSS schemes capable of computing on multi-source data either only support evaluating polynomials whose degrees are limited by the number of participants, or involve relatively complex multi-key HE in their constructions. In this paper, we introduce the formal notion of multi-key HSS, and propose a multi-key two-server HSS scheme from LWE without multi-key HE, by leveraging the technique of homomorphic linear combination that was previously used to construct multi-key FHE from GSW-FHE in [18]. As a result, our scheme supports server-aided secure computation of restricted multiplication straight-line (RMS) programs over private data from multiple independent sources, and the number of input clients and the degree of evaluated polynomials can be as high as a polynomial in the security parameter. Finally, we prove the security of the multi-key HSS scheme, and demonstrate its implementation performance.

Keywords: Homomorphic secret sharing · Learning with errors · Nearly linear decryption · Homomorphic linear combination

1 Introduction

Homomorphic secret sharing (HSS), first introduced by Boyle et al. [3], enables multiple non-interactive parties to homomorphically evaluate a public function in the setting of secret sharing without the use of somewhat/fully homomorphic encryption (S/FHE) [15,16,23]. More specifically, an (n, m)-HSS scheme allows n input clients to secretly share their private data $x_1, \dots, x_n$ among m computing servers. Each server is able to locally evaluate a partial result upon

The authors gratefully acknowledge the anonymous reviewers for their valuable comments and suggestions which improved the quality of this paper.

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 248–269, 2023.
https://doi.org/10.1007/978-3-031-35486-1_12

the computation task, and the final result of the distributed computation can be reconstructed by all partial results. Over recent years, HSS schemes designed for some specific function classes, such as branching programs and general circuits, have emerged [3,5,7,12,13,24,25]. HSS can replace S/FHE to achieve the same functionality in some scenarios, especially in secure multi-party computation (MPC), serving as a building block of some secure computation protocols [1,2,8,17].

An attractive application of HSS is private outsourcing computation. Although existing HSS schemes provide low-communication solutions to data privacy issues in such scenarios, they do not work well in the context of joint computation among multiple independent data sources. In the latter case, a public evaluation task is issued by an output client, involving the private data from a set of input clients. Since traditional encryption schemes (e.g., [3,13]) or S/FHE schemes (e.g., [12]) are required as the underlying construction of HSS schemes, a naive solution is to replace the underlying encryption schemes with the corresponding multi-key versions. However, this would cause the extended HSS schemes to remain at least as complex as their underlying multi-key encryption schemes and lose competitiveness compared to S/FHE. On the other hand, Boyle et al. [5] proposed an HSS scheme from (R)LWE without S/FHE, which is geared towards a competitive alternative to S/FHE. But it also focuses on the single-key setting, and no natural multi-key extension exists. The trusted setup is required in these HSS schemes to generate and distribute the same encrypted public key to input clients when there is more than one input client, which is a strong trust assumption. Till now, there is not yet a multi-key HSS scheme tailored for private outsourcing computation with multi-source data. Motivated by this gap, we manage to provide a feasible solution without complex cryptographic primitives or assumptions.

1.1 Our Contributions

In this paper, we first propose the formal definition of multi-key homomorphic secret sharing (MKHSS), and instantiate an n-client two-server MKHSS scheme, which is a multi-key extension of the HSS scheme in [5]. Our contributions are as follows:

- We propose a modified version of the Regev encryption scheme. The crucial differences are that the gadget vector is introduced in the ciphertext and the last component of the secret key vector of the Regev scheme is disclosed. Also, we prove the semantic security of our modified Regev scheme.
- We apply the homomorphic linear combination in [18] to our modified Regev encryption. Specifically, we modify the original matrix-matrix combination used for multi-key HE construction to a new combination of the matrix-vector form. The result is a modified algorithm that combines an arbitrary vector $\mathbf{v}$ and a modified Regev ciphertext $\mathbf{C}$ of a vector $\mathbf{r} \in \{0,1\}^m$ into a new "ciphertext" $\mathbf{C}_{lc}$ under the d-dimension secret key $\mathbf{s}$ such that either $\mathbf{s} \cdot \mathbf{C}_{lc} \approx (q/p) \cdot \mathbf{r} \cdot \mathbf{v}$ or $\mathbf{s} \cdot \mathbf{C}_{lc} \approx (q/p) \cdot \mathbf{s}[d] \cdot \mathbf{r} \cdot \mathbf{v}$ holds, where $\mathbf{s}[d]$ is the d-th component of $\mathbf{s}$.

- Utilizing the above two key components, we implement a share conversion procedure without ciphertext inflation, and construct an MKHSS scheme for the evaluation of Restricted Multiplication Straight-line (RMS) programs, where each client's private input is independently shared and homomorphically calculated between two non-colluding servers. The number of input clients and the degree of evaluated polynomials can be as high as a polynomial in the security parameter. Compared with HSS in [5], the extension only additionally consumes tolerable resources in terms of each client's input share and the server's computational times. To the best of our knowledge, this is the first attempt to construct multi-key HSS without using multi-key HE.

1.2 Related Work

The original $(1, 2)$-HSS scheme was proposed by Boyle et al. [3], which combines a threshold version of ElGamal with linear secret sharing to allow a compact evaluation of branching programs on input shares under the DDH assumption. However, a non-negligible error probability was introduced into the scheme due to the local share conversion algorithm, and could be further eased by servers with increasing their operating times.

Subsequently, Fazio et al. [13] extended the result in [3] and further constructed a $(1, 2)$-HSS scheme for branching programs based on the circular security of the Paillier encryption scheme. Besides, the scheme optimizes the former work by applying the "Las Vegas"-style share conversion protocols [4] as an alternative, which reduces the number of repetitions required to reach a reasonable overall error bound.

Based on the framework of HSS in [3], a $(1, 2)$-HSS construction from (R)LWE without S/FHE was proposed by Boyle et al. [5] and can be instantiated by any encryption scheme with nearly linear decryption property. Compared to the previous DDH-based schemes [3,13], this construction supports secure evaluation for the class of polynomial-size branching programs with a negligible error probability and superpolynomial-size plaintext space. Moreover, it offers cheaper homomorphic multiplications, a simpler setup procedure and no noise growth over S/FHE-based solutions for the same program class [1,11].

Chen et al. [7] introduced a verification process into [5] to propose a two-server verifiable HSS model, which aims to ensure the correctness of evaluation in the scenarios of outsourcing computation. Similarly, it supports the calculation of a function with the degree as high as a polynomial in security parameter.

The above schemes [3,5,7,13] are applicable to evaluating branching programs on multiple private inputs under the two-server model. Essentially, they are (1, 2)-HSS schemes, in other words, single-key HSS schemes. That is, the same key is required by multiple parties to generate corresponding input shares supporting subsequent calculations from their private data. Due to the underlying encryption schemes (e.g., ElGamal([14]), Paillier([10]), LPR([19]) etc.) and specific share conversion procedures involved in the construction of these HSS schemes, there is no natural extension to multi-key HSS schemes.

Lai et al. [12] relied on arbitrary degree-k homomorphic encryption scheme to construct a $(1, m)$-HSS scheme for evaluations of polynomials with degree $(k+1)m-1$, which mainly adopted the idea of [6]. Their construction can be naturally extended to any (n, m)-HSS under the multi-key setting by simply replacing the HE schemes with multi-key ones. However, the complexity of multi-key evaluations additionally depends on the multi-key S/FHE schemes involved, leading to no significant advantage in terms of efficiency over the multi-key S/FHE schemes. And the maximum degree of polynomials that can be evaluated depends on the underlying encryption scheme and the number of computing servers.

Tsaloli et al. [24] presented an instantiation of an n-client m-server multiplicative verifiable HSS scheme for multiplications of n $(n \leq m)$ elements under the hardness assumption of the fixed inversion problem in bilinear maps. The scheme inherently allows evaluations on data from different sources and the correctness of results can be publicly verified. In addition, only multiplications can be performed and hence the scheme is only available for limited application scenarios.

Both [12] and [24] are (n, m)-HSS schemes and can inherently realize the joint calculation of polynomials on multi-party private data without the share generation under the common key. Nevertheless, they suffered from restricted computing functions or the expensive computation overhead of high-degree polynomials, resulting in their lack of flexibility and being only applicable to limited outsourcing computing scenarios.

1.3 Organization

In Sect. 2 we introduce some notations and techniques that will be used in our construction. The formal definition of MKHSS is given in Sect. 3. Section 4 contains the detailed construction and analysis of our MKHSS. In Sect. 5 we present both implementation and performance analysis of our scheme. Finally, we draw conclusions in Sect. 6.

2 Preliminaries

Denote with $\lambda \in \mathbb{N}$ a security parameter, and use $poly(\lambda)$ to denote any function bounded by a polynomial in λ. Address any negligible function in λ with $negl(\lambda)$. O and ω denote asymptotic upper bound and non-asymptotically tight below bound respectively.

Denote column-vectors by bold lower-case letters and matrices by bold upper-case letters. For an l-dimensional vector $\mathbf{a}$ and $i \in \{1, \ldots, l\}$, $\mathbf{a}[i]$ refers to the i-th component of $\mathbf{a}$. For a distribution χ, $r \overset{\$}{\leftarrow} \chi$ means that r is chosen from χ uniformly at random. For a real number $x \in \mathbb{R}$, by $\lfloor x \rceil \in \mathbb{Z}$ denote the element closest to x, and by $|x|$ refer to the length of x. For a positive integer n, $[n]$ denotes the set $\{1, \ldots, n\}$. The inner product between two vectors is denoted by $\langle \mathbf{a}, \mathbf{b} \rangle$. For a matrix $\mathbf{X}$, $\mathbf{X}^\top$ denotes the transpose of $\mathbf{X}$. Let $\mathbf{A}[i, \cdot]$ be the i-th row of the matric $\mathbf{A}$ and $\mathbf{A}[\cdot, j]$ the j-th column. For matrices $\mathbf{A}$ and $\mathbf{B}$, $[\mathbf{A}\|\mathbf{B}]$ denotes the concatenation of $\mathbf{A}$ with $\mathbf{B}$. For $B \in \mathbb{N}$, define $[\mathbb{Z}]_B := \{x \in \mathbb{Z} | x \leq B\}$.

2.1 HSS from Encryption with Nearly Linear Decryption

Our construction of MKHSS is mainly based on the HSS scheme in [5], which supports the calculation of RMS programs over secret shares generated under the same key. The core technique is a public-key encryption with nearly linear decryption, which satisfies two key properties: encrypts certain key-dependent messages without the knowledge of the secret key; and distributively decrypts a ciphertext. Here is a brief review of this scheme.

The HSS scheme is parameterized by $p, q, B_{sk}, B_{ct} \in \mathbb{N}$, subject to $p|q$, $p \geq \lambda^{\omega(1)}$, $q/p \geq \lambda^{\omega(1)}$ and $B_{sk}, B_{ct} \leq poly(\lambda)$. Set a ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$ and $\mathcal{R}_p := \mathcal{R}/p\mathcal{R}$ are two quotient rings, where N is a power of 2. The secret key $\mathbf{s}$ satisfies $\|\mathbf{s}\|_\infty \leq B_{sk}$, and the error e satisfies $\|e\|_\infty \leq B_{ct}$.

Definition 1 (Encryption scheme with nearly linear decryption). *Let* $\mathsf{PKE} := (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be a public-key encryption scheme with pseudorandom ciphertexts and parameterized by* $(p, q, d, B_{sk}, B_{ct}, \mathcal{R})$. PKE *satisfies the property of nearly linear decryption if for any* $(pk, \mathbf{s})$ *in the image of* $\mathsf{PKE.Gen}(1^\lambda)$ *where* $\mathbf{s}$ *is of the form* $(1, \hat{\mathbf{s}})$ *for some* $\hat{\mathbf{s}} \in \mathcal{R}_p^{d-1}$, *for any message* $m \in \mathcal{R}_p$ *and for any ciphertext* $\mathbf{c} \in \mathcal{R}_q^d$ *in the image of* $\mathsf{PKE.Enc}(pk, m)$, *it further holds* $\langle \mathbf{s}, \mathbf{c} \rangle = (q/p) \cdot m + e \mod q$.

Definition 2 (KDM oracle). *Let* PKE *be a public-key encryption scheme with nearly linear decryption as defined above. There exists a* PPT *algorithm* $\mathsf{PKE.OKDM}$*:*

- $\hat{\mathbf{c}}_i \leftarrow \mathsf{PKE.OKDM}(pk, m, i)$*: On input a public key* pk*, a value* $m \in \mathcal{R}$ *and a sequence number* $i \in \{1, \ldots, d\}$*, the algorithm computes and outputs a ciphertext* $\hat{\mathbf{c}}_i := (q/p) \cdot m \cdot \mathbf{u}_i + \mathbf{c} \mod q$*, where* $(pk, \mathbf{s}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$*,* $\mathbf{c} \leftarrow \mathsf{PKE.Enc}(pk, 0)$ *and the i-th unit vector* $\mathbf{u}_i \in \mathcal{R}_q^d$*.*

$\mathsf{PKE.OKDM}$ *returns the componentwise encryption of* $m \cdot \mathbf{s}$ *without access to the secret key, serving as a key-dependent message oracle. And the following properties are required:*

Nearly linear decryption *to the message* $m \cdot s_i$*: For any ciphertext* $\hat{\mathbf{c}}_i \in \mathcal{R}_q^d$ *in the image of* $\mathsf{PKE.OKDM}(pk, m, i)$*, it holds that* $\langle \mathbf{s}, \hat{\mathbf{c}}_i \rangle = (q/p) \cdot (m \cdot s_i) + e \mod q$*.*
Security: *For any* PPT *adversary* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *winning holds that*

$$\mathsf{Adv}^{\mathsf{kdm-ind}}_{\mathsf{PKE.OKDM},\mathcal{A}}(\lambda) := \left| Pr\left[\mathcal{A}(pk) = \beta \;\middle|\; \begin{array}{c} (pk, sk) \leftarrow PKE.Gen(1^\lambda), \\ \beta \leftarrow \{0,1\}, \\ \beta' \leftarrow \mathcal{A}^{\mathcal{O}_{KDM}(\cdot,\cdot)}(pk) \end{array} \right] - \frac{1}{2} \right| \leq negl(\lambda),$$

where the oracle $\mathcal{O}_{\mathsf{KDM}}(m, i)$ *is defined as follows: if* $\beta = 0$*, it returns* $\mathsf{PKE.OKDM}(pk, m, i)$*; else, it returns* $\mathsf{PKE.Enc}(pk, 0)$*.*

Definition 3 (Distributed decryption of ciphertexts). *Let* PKE *be a public-key encryption scheme with nearly linear decryption as defined above. There exists a deterministic polynomial time algorithm* $\mathsf{PKE.DDec}$*:*

- $\hat{t}_b \leftarrow \mathsf{PKE.DDec}(b \in \{0,1\}, \mathbf{t}_b, \mathbf{c})$*: On input an index* $b \in \{0,1\}$*, a share* $\mathbf{t}_b \in \mathcal{R}_q^d$ *and a ciphertext* $\mathbf{c} \in \mathcal{R}_q^d$*, the algorithm computes and outputs a result* $\hat{t}_b := \lfloor (p/q) \cdot (\mathbf{t}_b \cdot \mathbf{c}) \rceil \mod p \mod q$*.*

Lemma 1 (Distributed decryption of sums of ciphertexts [5]). *Let* PKE *be a public-key encryption scheme with nearly linear decryption as defined above. For all* $x \in \mathcal{R}_p$*, for* $\mathbf{t}_0, \mathbf{t}_1 \xleftarrow{\$} \mathcal{R}_q^d$ *subject to* $\mathbf{t}_0 + \mathbf{t}_1 = x \cdot \mathbf{s} \mod q$*, for* $B_{add} \in \mathbb{N}$*, for all messages* $m_1, \ldots, m_{B_{add}} \in \mathcal{R}_p$ *with* $m := \sum_{i=1}^{B_{add}} m_i$*, and for all ciphertexts* $\mathbf{c}_i$ *of* m_i *with* $\mathbf{c} := \sum_{i=1}^{B_{add}} \mathbf{c}_i$*, it holds that* $\mathsf{PKE.DDec}(0, \mathbf{t}_0, \mathbf{c}) + \mathsf{PKE.DDec}(1, \mathbf{t}_1, \mathbf{c}) = x \cdot m \mod q$ *with probability of at least* $1 - N \cdot (N \cdot B_{add} \cdot \|x\|_\infty \cdot B_{ct} \cdot p/q + \|x \cdot m\|_\infty / p + p/q + 1/p) \geq 1 - \lambda^{-\omega(1)}$*.*

2.2 RMS Programs

Next, we introduce the Restricted Multiplication Straight-line (RMS) program, which is the computational model applied in [5]. It is a class of programs that captures all branching programs in particular logarithmic space computations and logarithmic depth circuits. And it can actually compute all functions.

Definition 4 (RMS programs [9]). *An RMS program is an arbitrary sequence of the first four of the following instructions bounded by* B_{max} *and terminates with an output instruction:*

load : *Load an input* a_i *as a memory value:* b_i $(id, b_i \leftarrow a_i)$*;*
$\mathsf{add1}$: *Add memory values* b_i *and* b_j $(id, b_k \leftarrow b_i + b_j)$*;*
$\mathsf{add2}$: *Add input values* a_i *and* a_j*:* $(id, a_k \leftarrow a_i + a_j)$*;*
mult : *Multiply memory value* b_i *by input* a_i*:* $(id, b_k \leftarrow a_i \cdot b_i)$*;*
output : *Output memory value* b_k*:* $(id, \hat{O}_j \leftarrow b_k)$*.*

All instructions above are sorted according to a unique identifier $id \in \mathcal{S}_{id}$*. During execution, the size of an intermediate computation value at any step remains "small" and is bounded by* B_{max}*, i.e.* $\|b\|_\infty \leq B_{max}$*, otherwise the program terminates and outputs* $\perp$*. As in [5], the maximum number of additions on input values is also denoted by* P_{inp+}*.*

3 Multi-key Two-Server Homomorphic Secret Sharing

In this section, we present the notion of multi-key homomorphic secret sharing in plain model. Our system consists of: a set of input clients, two computing servers and an output client. Note that our system model strictly requires that the output client cannot play the role of the input client, especially in the context of computations involving only two variables from two different input clients.

Definition 5 (Multi-key Homomorphic Secret Sharing (MKHSS)). *An n-input (1-output) 2-server multi-key homomorphic secret sharing scheme* $\mathsf{MKHSS} = (\mathsf{Gen}, \mathsf{Share}, \mathsf{Eval}, \mathsf{Rec})$ *for a class of programs $\mathcal{P}$ over $\mathbb{Z}$ with input space $\mathcal{I} \subseteq \mathbb{Z}$ consists of the following* PPT *algorithms/protocols:*

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*: On input the security parameter 1^λ, the key generation algorithm outputs a public key pk and a secret key sk.*
- $(s_i^{(0)}, s_i^{(1)}) \leftarrow \mathsf{Share}(i, pk, sk, x)$*: Given an input index $i \in [n]$, a public key pk, a secret key sk, and a value $x \in \mathcal{I}$, the sharing algorithm outputs a pair of input shares $(s_i^{(0)}, s_i^{(1)})$.*
- $y_b \leftarrow \mathsf{Eval}(b, P, \{s_i^{(b)}\}_{i\in[n]}, r)$*: The server $\mathcal{S}_b$ executes the evaluation protocol on inputs an index $b \in \{0,1\}$, a program $P \in \mathcal{P}$ with n input values, the corresponding tuple of shares $\{s_i^{(b)}\}_{i\in[n]}$, and an integer $r \geq 2$. Upon termination, the server $\mathcal{S}_b$ outputs the corresponding output share y_b.*
- $y \leftarrow \mathsf{Rec}(y_0, y_1)$*: On input a pair of output shares (y_0, y_1), the reconstruction algorithm outputs the result y of the evaluation.*

MKHSS should satisfy the following correctness, security and context hiding: **Correctness**. The correctness of a multi-key HSS scheme ensures that the outputs originate from honest computing servers can be reconstructed to the accurate computing results.

Definition 6 (Correctness). *An n-input 2-server multi-key HSS scheme for a class of programs $\mathcal{P}$ is correct if for any security parameter $\lambda \in \mathbb{N}$, for any n-tuple of inputs $(x_1, \ldots, x_n) \in \mathcal{I}^n$, for any program $P \in \mathcal{P}$ with size $|P| \leq poly(\lambda)$ and $P(x_1, \ldots, x_n) \neq \perp$, for any integer $r \geq 2$, for $\forall i \in [n]$, $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$ and $(s_i^{(0)}, s_i^{(1)}) \leftarrow \mathsf{Share}(i, pk_i, sk_i, x_i)$ it holds that*

$$\mathsf{Pr}^{\mathsf{cor}}_{\mathsf{MKHSS},\{x_i\}_{i\in[n]},P,r}(\lambda) := \mathsf{Pr}[y_0 + y_1 = P(x_1, \ldots, x_n) \mod r] \geq 1 - negl(\lambda),$$

where $y_b \leftarrow \mathsf{Eval}(b, P, \{s_i^{(b)}\}_{i\in[n]}, r)$ for $b \in \{0,1\}$. The scheme is perfectly correct if the above probability is exactly 1. Further, if there exists a polynomial $\theta(\cdot)$, such that $|y_b| \leq \theta(\lambda)$, the multi-key HSS scheme is compact.

Security. The security of a multi-key HSS scheme guarantees that no information about private inputs is disclosed to any individual server. It preserves the privacy of input clients against computing servers.

Definition 7 (Security). *An n-input 2-server multi-key HSS scheme is secure if for any security parameter $\lambda \in \mathbb{N}$, there exists a negligible function $negl(\lambda)$ such that for any* PPT *adversary $\mathcal{A}$ that on input 1^λ output a bit $b \in \{0,1\}$, and $x_0, x_1 \in \mathcal{I}$, the advantage of $\mathcal{A}$ holds that*

$$\mathsf{Adv}^{\mathsf{sec}}_{\mathsf{MKHSS},\mathcal{A}}(\lambda) := \left| \mathsf{Pr}\left[\mathcal{A}(input_b) = \beta \;\middle|\; \begin{array}{c} (b, x_0, x_1, state) \\ \leftarrow \mathcal{A}(1^\lambda), \\ \beta \leftarrow \{0,1\}, \\ (pk, sk) \leftarrow \\ \mathsf{MKHSS.Gen}(1^\lambda), \\ (s_i^{(0)}, s_i^{(1)}) \leftarrow \\ \mathsf{MKHSS.Share} \\ (i, pk, sk, x_\beta), \\ input_b := (state, pk, s_i^{(b)}) \end{array} \right] - \frac{1}{2} \right| \leq negl(\lambda).$$

Context Hiding. Context hiding assures that the output client learns nothing beyond the result of evaluation, which preserves the privacy of input clients against the output client. And this property makes sense for our system model, where the output client is actually different from the input clients.

Definition 8 (Context Hiding). *An n-input 2-server multi-key HSS scheme is context-hiding if for any security parameter $\lambda \in \mathbb{N}$, for any program $P \in \mathcal{P}$ with size $|P| \leq poly(\lambda)$ and $P(x_1, \ldots, x_n) \neq \perp$, there exists a negligible function $negl(\lambda)$ and a* PPT *simulator $\mathcal{S}$ such that for any* PPT *adversary $\mathcal{A}$ that on input 1^λ output $x_1, \ldots, x_n \in \mathcal{I}$, the advantage of $\mathcal{A}$ holds that*

$$\mathsf{Adv}^{\mathsf{ch}}_{\mathsf{MKHSS},\mathcal{S},\mathcal{A}}(\lambda) := \left| \mathsf{Pr}\left[\mathcal{A}(input_\beta) = \beta \;\middle|\; \begin{array}{c} (x_1, \ldots, x_n, r, state) \leftarrow \mathcal{A}(1^\lambda), \\ \beta \leftarrow \{0,1\}, \\ (pk_i, sk_i) \leftarrow \\ \mathsf{MKHSS.Gen}(1^\lambda) \forall i \in [n], \\ if \beta = 0, (s_i^{(0)}, s_i^{(1)}) \leftarrow \\ \mathsf{MKHSS.Share} \\ (i, pk_i, sk_i, x_i) \forall i \in [n], \\ \wedge y_0^{(j)} \leftarrow \mathsf{MKHSS.Eval} \\ (j, P, \{s_i^{(j)}\}_{i \in [n]}, r) \forall j \in \{0,1\}; \\ else, (y_1^{(0)}, y_1^{(1)}) \leftarrow \\ \mathcal{S}(1^\lambda, P(x_1, \ldots, x_n), r), \\ input_\beta := (state, (y_\beta^{(0)}, y_\beta^{(1)})) \end{array} \right] - \frac{1}{2} \right| \leq negl(\lambda).$$

4 The Construction of Multi-key HSS

Inspired by [5], we now present our n-input (1-output) 2-server multi-key homomorphic secret sharing scheme MKHSS for a class of RMS programs $\mathcal{P}$. We first modify the Regev encryption scheme, by which each input client generates a pair of shares for individual private input and distributes the input shares between two computing servers. Then by leveraging the technique of homomorphic linear combination in [18], we show a method for the computing servers to evaluate a function over inputs from all input clients with different keys. Finally, the output client reconstructs the evaluation result from partial results returned by both servers.

4.1 The Modified Regev Encryption

To construct our multi-key HSS scheme, we present a modified version of the standard Regev encryption [22] in which: (1) the gadget vector $\mathbf{g}^\top$ is introduced in the ciphertexts, and (2) the last component of the secret key is public.

We claim that for $l := \lfloor \log q \rfloor + 1$, the gadget vector $\mathbf{g}^\top = (1, 2, \ldots, 2^{l-1})$ and the function $\mathbf{g}^{-1} : \mathbb{Z}_q \rightarrow \mathbb{Z}^l$ such that $\mathbf{x} \leftarrow \mathbf{g}^{-1}(u)$ is subgaussian with $O(1)$ and satisfies $\langle \mathbf{g}, \mathbf{x} \rangle = u$, as introduced in [20].

The modified Regev encryption scheme consists of the following PPT algorithms $\mathsf{MRegev} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$:

- $\mathsf{MRegev.Setup}(1^{\lambda}) \to params$: On the security parameter λ, choose the modulus value q, the lattice dimension d, the secret-key distribution χ_{sk} and the error distribution χ. Let $m = O(d \log q)$ be the number of LWE instances. The setup algorithm outputs $params = (q, d, m, \chi_{sk}, \chi)$.
- $\mathsf{MRegev.KGen}(params) \to (pk, sk)$: Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times (d-1)}$, $\hat{\mathbf{s}} \xleftarrow{\$} \chi_{sk}^{d-1}$ and $\mathbf{e} \xleftarrow{\$} \chi^m$. Compute $\mathbf{b} = \mathbf{A} \cdot \hat{\mathbf{s}} + \mathbf{e} \mod q$. Set $\mathbf{s} = (1, \hat{\mathbf{s}})$ and $\mathbf{P} = [\mathbf{b} \| -\mathbf{A}] \in \mathbb{Z}_q^{m \times d}$. The key generation algorithm outputs a secret key $sk = \mathbf{s}$ and a public key $pk = (\mathbf{P}, \mathbf{s}[d])$.
- $\mathsf{MRegev.Enc}(pk, \mu, i \in \{0,1\}) \to \mathbf{C}$: Given a message $\mu \in \mathbb{Z}_p$, if the index $i = 0$, let $\mathbf{M}^{\top} = \begin{bmatrix} \mu\, 0 \cdots 0 \\ 0\; 0 \cdots 0 \end{bmatrix} \in \mathbb{Z}_p^{2 \times d}$; else, let $\mathbf{M}^{\top} = \begin{bmatrix} 0 \cdots 0\; 0 \\ 0 \cdots 0\; \mu \end{bmatrix} \in \mathbb{Z}_p^{2 \times d}$. Sample $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m \times 2l}$. The encryption algorithm outputs a ciphertext $\mathbf{C} = \mathbf{P}^{\top}\mathbf{R} + (q/p)\mathbf{M} \otimes \mathbf{g} \in \mathbb{Z}_q^{d \times 2l}$.
- $\mathsf{MRegev.Dec}(sk, \mathbf{C}, i \in \{0,1\}) \to \mu$: Given a ciphertext $\mathbf{C} \in \mathbb{Z}_q^{d \times 2l}$, if the index $i = 0$, set $u = \lfloor \frac{q}{p} \rfloor^{-1} \mod q$, and define a vector $\mathbf{u}^{\top} = (\mathbf{g}^{-1}(u), 0, \cdots, 0)$ $\in \mathbb{Z}_q^{2l}$; else, set $u = (\lfloor \frac{q}{p} \rfloor \mathbf{s}[d])^{-1} \mod q$, and define a vector $\mathbf{u}^{\top} = (0, \cdots, 0, \mathbf{g}^{-1}(u)) \in \mathbb{Z}_q^{2l}$. Compute and output the decrypted message $\mu = sk \cdot \mathbf{C} \cdot \mathbf{u}$.

Correctness of Decryption. In the case of the index $i = 1$, we have

$$\begin{aligned} \mu = sk \cdot \mathbf{C} \cdot \mathbf{u} &= (1, \hat{\mathbf{s}})(\begin{bmatrix} \mathbf{b} \\ -\mathbf{A} \end{bmatrix} \mathbf{R} + (q/p) \begin{bmatrix} 0 \cdots 0\; 0 \\ 0 \cdots 0\; \mu \end{bmatrix}^{\top} \otimes \mathbf{g})(0, \cdots, 0, \mathbf{g}^{-1}(u))^{\top} \\ &= \mathbf{e} \cdot \mathbf{R}(0, \cdots, 0, \mathbf{g}^{-1}(u))^{\top} + (q/p)(1, \hat{\mathbf{s}})(0, \cdots, 0, \mu \cdot u)^{\top} \\ &\approx (q/p)\mathbf{s}[d] \cdot \mu \cdot u \\ &\approx \mu. \end{aligned}$$

Also, in the case of the index $i = 0$, the correctness of decryption is held by the same token. □

Before giving a security proof of the modified Regev encryption scheme, we introduce the following lemma.

Lemma 2 ([22]). Let the $\mathsf{LWE}_{\mathsf{n,q},\chi}$ assumption holds with $params = \{n, q, \chi, m\}$. Then, for $m = O(n \log q)$, $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m \times 2l}$, and $\mathbf{P}$ as generated above, the joint distribution $(\mathbf{P}^{\top}, \mathbf{P}^{\top}\mathbf{R})$ is computational indistinguishable from uniform over $\mathbb{Z}_q^{d \times m} \times \mathbb{Z}_q^{d \times 2l}$.

Then the semantic security of the above scheme is summarized in the following theorem.

Theorem 1. *The above modified Regev encryption scheme* $\mathsf{MRegev}_{\mathsf{q,d,m},\chi_{\mathsf{sk}},\chi}$ *is semantically secure if the* $\mathsf{Regev}_{\mathsf{q,d-1,m},\chi_{\mathsf{sk}},\chi}$ *encryption scheme is secure.*

With the limitation in space, here we give the sketch of the proof.

Proof. First, we define another encryption scheme MRegev', where the only difference from Regev is that its encryption algorithm $\mathsf{MRegev}'.\mathsf{Enc}$ outputs

$\mathbf{C} = \mathbf{P}^\top \mathbf{R} + (q/p)\mathbf{M} \otimes \mathbf{g}$ as that in $\mathsf{MRegev.Enc}$. Due to the columns of $\mathbf{P}^\top \mathbf{R}$ are simply Regev encryptions of 0 for dimension d, the semantic security of MRegv' follows directly from Lemma 2. Thus, $\mathsf{MRegev}'_{\mathsf{q,d},\chi_{\mathsf{sk}},\chi}$ is semantically secure if $\mathsf{Regev}_{\mathsf{q,d},\chi_{\mathsf{sk}},\chi}$ is.

Then we define MRegev'', the encryption scheme which is almost the same as MRegev' except that: (1) its public key contains an additional vector $\mathbf{a}$ randomly chosen from $\mathbb{Z}_q^m$, i.e. $pk = (\mathbf{P}, \mathbf{a})$; and (2) its encryption algorithm $\mathsf{MRegev}''.\mathsf{Enc}$ outputs $\mathbf{C} = \mathbf{P}^\top \mathbf{R} + (q/p)\mathbf{M} \otimes \mathbf{g}$ along with $\mathbf{C}' = \mathbf{a}^\top \mathbf{R}$. Obviously, the distribution of $\mathbf{a}^\top \mathbf{R}$ is statistically indistinguishable with the uniform distribution. Therefore, $\mathsf{MRegev}''_{\mathsf{q,d,m},\chi_{\mathsf{sk}},\chi}$ is semantically secure if $\mathsf{MRegev}'_{\mathsf{q,d-1,m},\chi_{\mathsf{sk}},\chi}$ is.

Now suppose there exists a PPT adversary $\mathcal{A}$ which breaks the security of $\mathsf{MRegev}_{\mathsf{q,d,m},\chi_{\mathsf{sk}},\chi}$ with non-negligible advantages ϵ, another PPT adversary $\mathcal{B}$ against the semantic security of $\mathsf{MRegev}''_{\mathsf{q,d,m},\chi_{\mathsf{sk}},\chi}$ with non-negligible advantages $\frac{1}{2}\epsilon$ can be constructed as follows. The challenger $\mathcal{C}$ participates in the challenge game of MRegev'' and receives the parameters $\{q, d, m, \chi_{sk}, \chi\}$ along with the public key $(\mathbf{P}, \mathbf{a}) = ([\mathbf{b}\| -\mathbf{A}], \mathbf{a})$ which are forwarded to $\mathcal{A}$. $\mathcal{A}$ chooses an arbitrary vector $\mathbf{s}' \in \chi_{sk}$ and correctly yields a public key $\mathbf{P}' = [\mathbf{b} + \mathbf{a}^\top \cdot \mathbf{s}' \| -\mathbf{A} \| -\mathbf{a}] \in \mathbb{Z}_q^{m \times d}$. $\mathcal{A}$ then sends the parameters $\{q, d, m, \chi_{sk}, \chi\}$, $\mathbf{P}'$ and $\mathbf{s}'$ to $\mathcal{B}$. $\mathcal{B}$ randomly chooses two distinct messages m_0,$m_1 \in \mathbb{Z}_q$ such that $m_0 \neq m_1$ and returns them to $\mathcal{A}$. $\mathcal{A}$ transfers the two messages to $\mathcal{C}$. $\mathcal{C}$ randomly chooses a bit $b \in \{0, 1\}$ and encrypts m_b as $(\mathbf{C}, \mathbf{C}')$ and $(\hat{\mathbf{C}}, \mathbf{C}')$ by invoking $\mathsf{MRegev}''.\mathsf{Enc}$, where $\mathbf{C} = \mathsf{MRegev}''.\mathsf{Enc}(\mathbf{P}, m_b, 0) = \mathbf{P}^\top \mathbf{R} + (q/p)\mathbf{M}_b \otimes \mathbf{g}$, $\hat{\mathbf{C}} = \mathsf{MRegev}''.\mathsf{Enc}(\mathbf{P}, m_b, 1) = \mathbf{P}^\top \mathbf{R} + (q/p)\mathbf{M}'_b \otimes \mathbf{g}$ and $\mathbf{C}' = \mathbf{a}^\top \mathbf{R}$. The ciphertexts are forwarded to $\mathcal{B}$. $\mathcal{B}$ randomly chooses a bit $b' \in \{0, 1\}$, sets $\mathbf{C}[1, \cdot] = \mathbf{C}[1, \cdot] + \mathbf{s}'\mathbf{C}'$ and $\hat{\mathbf{C}}[1, \cdot] = \hat{\mathbf{C}}[1, \cdot] + \mathbf{s}'\mathbf{C}'$. $\mathcal{B}$ returns $\mathbf{C}^* = \begin{bmatrix} \mathbf{C} \\ \mathbf{C}' \end{bmatrix}$ and $\hat{\mathbf{C}}^* = \begin{bmatrix} \hat{\mathbf{C}} - (q/p)\mathbf{M}'_{b'} \otimes \mathbf{g} \\ \mathbf{C}' + (q/p)[0\|m_{b'}] \otimes \mathbf{g} \end{bmatrix}$ to $\mathcal{A}$. Finally, $\mathcal{B}$ feedbacks $\mathcal{A}$'s output to the challenger.

By construction above, if $b' = b$, $\mathcal{B}$ simulates the MRegev'' game for $\mathcal{A}$ faithfully, i.e. the ciphertexts forwarded to $\mathcal{A}$ are identical to that in MRegev. Thus, the success probability of $\mathcal{B}$ is identical to that of $\mathcal{A}$ breaking the security of MRegev. Else, if $b' \neq b$, the probability of $\mathcal{B}$ winning is at least $\frac{1}{2}$ in that the ciphertexts obtained by $\mathcal{A}$ are not correctly generated. Thus,

$$\begin{aligned}
&\mathsf{Pr}[\mathcal{B} \quad wins] = \mathsf{Pr}[\mathcal{B} \quad wins | b' = b] + \mathsf{Pr}[\mathcal{B} \quad wins | b' \neq b] \\
\geq\ &\frac{1}{2}\mathsf{Pr}[\mathcal{A} \quad wins] + \frac{1}{2} \times \frac{1}{2} = \frac{1}{2} \times (\frac{1}{2} + \epsilon) + \frac{1}{2} \times \frac{1}{2} = \frac{1}{2} + \frac{1}{2}\epsilon.
\end{aligned}$$

Consider the security of MRegev'', we conclude that the advantage of $\mathcal{A}$ winning is negligible. □

4.2 The Homomorphic Linear Combination for the Modified Regev Encryption

We leverage the idea of homomorphic linear combination for the construction of multi-key HE in [18], which is a generalization of that in [21], to design a similar homomorphic operation for the modified Regev encryption proposed above.

Precisely, we modify the homomorphic linear combination of matrix-matrix to that of matrix-vector, which becomes another key component in our main construction.

The polynomial-time deterministic algorithms $\mathsf{MRegev.HLC}$ describes the homomorphic linear combination for the modified Regev encryption, and is defined as follows:

- $\mathsf{MRegev.HLC}(\{\mathbf{C}^{(j)}\}_{j\in[m]}, \mathbf{v}, i \in \{0,1\}) \to \mathbf{t}$: On input a sequence of MRegev ciphertexts $\{\mathbf{C}^{(j)}\}_{j\in[m]}$ and an arbitrary vector $\mathbf{v} \in \mathbb{Z}_q^m$, if the given index $i = 0$, set $\mathbf{v}^{(j)} = (\mathbf{g}^{-1}(\mathbf{v}[j]), 0, \cdots, 0) \in \mathbb{Z}_q^{2l}$ for $j \in [m]$; else, set $\mathbf{v}^{(j)} = (0, \cdots, 0, \mathbf{g}^{-1}(\mathbf{v}[j])) \in \mathbb{Z}_q^{2l}$ for $j \in [m]$. Compute $\mathbf{t} = \sum_{j=1}^{m} \mathbf{C}^{(j)}\mathbf{v}^{(j)}$. The combined output is $\mathbf{t}$.

Theorem 2. *For $j \in [m]$, let $\mathbf{T}^{(j)} \in \mathbb{Z}_q^{d\times 2l}$ be a ciphertext of* $\mathsf{MRegev.Enc}(pk, \mathbf{r}[j], 0)$ *under a secret key* $\mathbf{s}$*, and $\hat{\mathbf{T}}^{(j)} \in \mathbb{Z}_q^{d\times 2l}$ be a ciphertext of* $\mathsf{MRegev.Enc}(pk, \mathbf{r}[j], 1)$ *under the same* $\mathbf{s}$*, where* $\mathbf{r} \in \{0,1\}^m$*. Let* $\mathbf{t}_{lc} \leftarrow \mathsf{MRegev.HLC}(\{\mathbf{T}^{(j)}\}_{j\in[m]}, \mathbf{v}, 0)$ *and* $\hat{\mathbf{t}}_{lc} \leftarrow \mathsf{MRegev.HLC}(\{\hat{\mathbf{T}}^{(j)}\}_{j\in[m]}, \mathbf{v}, 1)$*. Then* $\mathbf{s} \cdot \mathbf{t}_{lc} \approx (q/p) \cdot \mathbf{r} \cdot \mathbf{v}$ *and* $\mathbf{s} \cdot \hat{\mathbf{t}}_{lc} \approx (q/p) \cdot \mathbf{s}[d] \cdot \mathbf{r} \cdot \mathbf{v}$ *are satisfied respectively.*

Proof. In the case of the index $i = 1$, we have

$$\begin{aligned}
&\mathbf{s} \cdot \hat{\mathbf{t}}_{lc} = (1, \hat{\mathbf{s}}) \cdot \sum_{j=1}^{m} \hat{\mathbf{T}}^{(j)}\mathbf{v}^{(j)} \\
&= \sum_{j=1}^{m} (1, \hat{\mathbf{s}}) \cdot (\mathbf{P}^\top \mathbf{R} + (q/p) \begin{bmatrix} 0 & \cdots & \cdots & 0 \\ 0 & \cdots & 0 & \mathbf{r}[j] \end{bmatrix}^\top \otimes \mathbf{g})(0, \cdots, 0, \mathbf{g}^{-1}(\mathbf{v}[j]))^\top \\
&= \sum_{j=1}^{m} \mathbf{eR}(0, \cdots, 0, \mathbf{g}^{-1}(\mathbf{v}[j]))^\top + (q/p) \sum_{j=1}^{m} \mathbf{s}[d] \cdot \mathbf{r}[j] \cdot \mathbf{v}[j] \\
&\approx (q/p) \cdot \mathbf{s}[d] \cdot \mathbf{r} \cdot \mathbf{v}.
\end{aligned}$$

Similarly, the proof for the other case, i.e. the index $i = 0$, can be easily obtained by the same way. □

4.3 Our Construction

Our construction of multi-key HSS scheme takes advantage of the idea of homomorphic linear combination from [18] and the nearly linear decryption property in Regev encryption. It extends the single-key HSS in [5] to an HSS scheme applicable to independent multi-client scenarios. Our construction supports 5 homomorphic RMS program instructions. Now we present the main construction of the multi-key HSS in details.

$\underline{\mathsf{MKHSS.Gen}(1^\lambda)}$:

- Invoke $params \leftarrow \mathsf{MRegev.Setup}(1^\lambda)$.
- Invoke $(pk, sk) \leftarrow \mathsf{MRegev.KGen}(params)$ and output the key pair (pk, sk).

$\underline{\mathsf{MKHSS.Share}(k, pk_k, sk_k, x)}$:

- Parse $sk_k = \mathbf{s}_k := (1, \hat{\mathbf{s}}_k) \in \mathbb{Z}_q \times \mathbb{Z}_q^{d-1}$ and $pk_k = (\mathbf{P}_k, \mathbf{s}_k[d]) := ((\mathbf{b}_k||-\mathbf{A}_k), \mathbf{s}_k[d]) \in \mathbb{Z}_q^{m\times d} \times \mathbb{Z}_q$.
- Draw a PRF key $K_k \leftarrow \mathcal{K}$.
- Choose $ek_k^{(0)} \leftarrow \mathbb{Z}_q^d$ at random. Define $ek_k^{(1)} := sk_k - ek_k^{(0)} \mod q$.
- For $j \in [d]$, compute $\mathbf{c}_k^{(x,j)} \leftarrow \mathsf{Regev.OKDM}(pk_k, x, j) \in \mathbb{Z}_q^d$, where the random vector used for j-th encryption is $\mathbf{r}_k^{(j)} \in \{0,1\}^m$ and $\mathbf{e}_j \in \mathbb{Z}_q^d$ is the j-th unit vector. The encryption of input x is $\mathbf{C}_k^{(x)} = \{\mathbf{c}_k^{(x,j)}\}_{j\in[d]} \in \mathbb{Z}_q^{d\times d}$.
- For $i \in [d]$ and $j \in [m]$, invoke $\mathbf{T}_k^{(i,j)} \leftarrow \mathsf{MRegev.Enc}(pk_k, \mathbf{r}_k^{(i)}[j], 0)$, where $\mathbf{R}_k^{(i,j)} = [\mathbf{R}_k^{(i,j')}||\mathbf{R}_k^{(i,j'')}] \in \{0,1\}^{m\times l} \times \{0,1\}^{m\times l}$ is the randomly chosen matrix used for each encryption. Also invoke $\hat{\mathbf{T}}_k^{(i,j)} \leftarrow \mathsf{MRegev.Enc}(pk_k, \mathbf{r}_k^{(i)}[j], 1)$, where $\mathbf{R}_k^{(i,j''')} \in \{0,1\}^{m\times l}$ and the random matrix used for each encryption $\hat{\mathbf{R}}_k^{(i,j)} = [\mathbf{R}_k^{(i,j''')}||\mathbf{R}_k^{(i,j')}]$.
- Obtain a pair of input shares $s_k^{(b)} := (ek_k^{(b)}, K_k, \mathbf{C}_k^{(x)}, \{\mathbf{T}_k^{(i,j)}, \hat{\mathbf{T}}_k^{(i,j)}\}_{i\in[d],j\in[m]})$ for $b \in \{0,1\}$.

$\underline{\mathsf{MKHSS.Eval}(b, \{pk_i\}_{i\in[n]}, P, \{s_i^{(b)}\}_{i\in[n]}, r)}$:

For $i \in [n]$:

- Parse $pk_i = (\mathbf{P}_i, \mathbf{s}_i[d]) := ((\mathbf{b}_i||-\mathbf{A}_i), \mathbf{s}_i[d])$ and $s_i^{(b)} := (ek_i^{(b)}, K_i, \mathbf{C}_i^{(x)}, \{\mathbf{T}_i^{(k,j)}, \hat{\mathbf{T}}_i^{(k,j)}\}_{k\in[d],j\in[m]})$. For $l \in [m]$, compute $z_i^{(l)} = \sum_{j\in[n]\setminus\{i\}}(b_j[l] - b_i[l])\delta$, where $\delta = \frac{p}{q\sum_{j\in[n]}(\mathbf{s}_j[d]-1)}$. $\mathbf{z}_i := (z_i^{(1)}, \ldots, z_i^{(m)}) \in \mathbb{Z}_q^m$.
- For $k \in [d]$, invoke $\mathbf{t}_i^{(k)} \leftarrow \mathsf{MRegev.HLC}(\{\mathbf{T}_i^{(k,j)}\}_{j\in[m]}, \mathbf{z}_i, 0)$ and $\hat{\mathbf{t}}_i^{(k)} \leftarrow \mathsf{MRegev.HLC}(\{\hat{\mathbf{T}}_i^{(k,j)}\}_{j\in[m]}, \mathbf{z}_i, 1)$.
- For $j \in [d]$, compute $\overline{\mathbf{c}}^{(x,j)} = \mathbf{c}_i^{(x,j)} - \mathbf{t}_i^{(j)} + \hat{\mathbf{t}}_i^{(j)}$. The extended input share is $\overline{\mathbf{C}}^{(x)} = \{\overline{\mathbf{c}}^{(x,j)}\}_{j\in[d]} \in \mathbb{Z}_q^{d\times d}$ and the corresponding extended evaluation key $\overline{ek}_b = \sum_{i\in[n]} ek_i^{(b)}$.
- Parse P as a sequence of RMS operations and proceed as follows:
  (1) Load upon instruction $(id, \overline{\mathbf{C}}^{(x)})$: compute $t_b^{(x,j)} := \mathsf{MRegev.DDec}(b, \overline{ek}_b, \overline{\mathbf{c}}^{(x,j)}) + (1-2b)\cdot\sum_{i=1}^n \mathsf{PRF}(K_i, id) \mod q$ for $j \in [d]$. Then the componentwise memory value $\mathbf{t}_b^{(x)} := \{t_b^{(x,j)}\}_{j\in[d]} \in \mathbb{Z}_q^d$.
  (2) $\mathsf{Add1}$ upon instruction $(id, t_b^{(x)}, t_b^{(x')})$: compute $\mathbf{t}_b^{(x+x')} \leftarrow \mathbf{t}_b^{(x)} + \mathbf{t}_b^{(x')} + (1-2b)\cdot\sum_{i=1}^n \mathsf{PRF}(K_i, id) \mod q$.
  (3) $\mathsf{Add2}$ upon instruction $(id, \overline{\mathbf{C}}^{(x)}, \overline{\mathbf{C}}^{(x')})$: compute $\overline{\mathbf{C}}^{(x+x')} \leftarrow \overline{\mathbf{C}}^{(x)} + \overline{\mathbf{C}}^{(x')} \mod q$.
  (4) Mult upon instruction $(id, \mathbf{t}_b^{(x)}, \overline{\mathbf{C}}^{(x')})$: for $j \in [d]$, compute $t_b^{(x\cdot x',j)} := \mathsf{MRegev.DDec}(b, t_b^{(x,j)}, \overline{\mathbf{c}}^{(x',j)}) + (1-2b)\cdot\sum_{i=1}^n \mathsf{PRF}(K_i, id) \mod q$. Then the componentwise product $\mathbf{t}_b^{(x\cdot x')} := \{t_b^{(x\cdot x',j)}\}_{j\in[d]} \in \mathbb{Z}_q^d$.

(5) Output over $\mathbb{Z}_r$ upon instruction $(id, \mathbf{t}_b^{(x)})$: parse $\mathbf{t}_b^{(x)} := (x_b, \hat{\mathbf{t}}_b^{(x)})$ for some $x_b \in \mathbb{Z}_q$ and $\hat{\mathbf{t}}_b^{(x)} \in \mathbb{Z}_q^{d-1}$. Output $y_b = \frac{x_b}{n} \mod r$.

$\underline{\mathsf{MKHSS.Rec}(y_0, y_1)}$:

- Output the computing result of P as $y = y_0 + y_1$.

Let $\mathsf{PRF} : \mathcal{K} \times \mathcal{S}_{id} \to \mathbb{Z}_q^d$ be a pseudorandom function. The scheme is parameterized by module values $q, p \in \mathbb{N}$, an integer lattice dimension d, the number of LWE instances m, and distributions χ_{sk} over $\mathbb{Z}_p$ and χ over $\mathbb{Z}$ bounded by B_{sk} and B_{err} respectively, where $p|q$, $q/p \geq \lambda^{\omega(1)}$ and $d, m \leq poly(\lambda)$. The noise bound $B_{ct} \leq poly(\lambda)$. Let $B_{inp} \in \mathbb{N}$ with $p/B_{inp} \geq \lambda^{\omega(1)}$ and $q/(B_{inp} \cdot p) \geq \lambda^{\omega(1)}$.

Theorem 3. *The scheme* $\mathsf{MKHSS} = (\mathsf{Gen}, \mathsf{Share}, \mathsf{Eval}, \mathsf{Rec})$ *given above is an n-input 2-server multi-key homomorphic secret sharing scheme with input space $[\mathbb{Z}]_{B_{inp}}$ for the class of RMS programs with magnitude bound B_{max}, where $p/B_{max} \geq \lambda^{\omega(1)}$ and $q/(B_{max} \cdot p) \geq \lambda^{\omega(1)}$. More precisely, our* MKHSS *satisfies:*

- ***Correctness**. For any $\lambda \in \mathbb{N}$, for any $x_1, \ldots, x_n \in [\mathbb{Z}]_{B_{inp}}$, for any polynomial-sized RMS program P with $P(x_1, \ldots, x_n) \neq \perp$ and magnitude bound B_{max}, and P has maximum number of input addition instructions P_{inp+}, for any integer $2 \leq r \leq B_{max} \ll p$, it holds that* $\mathsf{Pr}^{\mathsf{cor}}_{\mathsf{MKHSS}, \{x_i\}_{i\in[n]}, P, r}(\lambda) \geq 1 - negl(\lambda)$.
- ***Security**. For any* PPT *adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ winning holds that* $\mathsf{Adv}^{\mathsf{sec}}_{\mathsf{MKHSS}, \mathcal{A}}(\lambda) \leq negl(\lambda)$.
- ***Context Hiding**. For any* PPT *adversary $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{S}$ such that* $\mathsf{Adv}^{\mathsf{ch}}_{\mathsf{MKHSS}, \mathcal{S}, \mathcal{A}}(\lambda) \leq negl(\lambda)$.

Obviously, our MKHSS scheme is mainly constructed from the modified Regev scheme with the properties of homomorphic linear combination and nearly linear decryption. Next we give the proof of Theorem 3.

Lemma 3 (Nearly Linear Decryption for Extended Ciphertext). *Let $\overline{\mathbf{C}}^{(x)} = \{\overline{\mathbf{c}}^{(x,j)}\}_{j\in[d]} = \{\mathbf{c}_k^{(x,j)} - \mathbf{t}_k + \hat{\mathbf{t}}_k\}_{j\in[d]} \in \mathbb{Z}_q^{d\times d}$ be the extended input share which corresponds to the k-th client's input, and $\overline{ek}_b = \sum_{i\in[n]} ek_i^{(b)} \in \mathbb{Z}_q^d$ for $b \in \{0, 1\}$ the corresponding extended evaluation key evaluated in* $\mathsf{MKHSS.Eval}$, *then we have $\overline{\mathbf{C}}^{(x)}$ is a valid ciphertext satisfying nearly linear decryption under the secret key $(\overline{ek}_0 + \overline{ek}_1)$.*

Proof. For $j \in [d]$,

$$\begin{aligned}\langle \overline{ek}_0 + \overline{ek}_1, \overline{\mathbf{c}}^{(x,j)} \rangle &= \langle \sum_{i\in[n]} (ek_i^{(0)} + ek_i^{(1)}), \overline{\mathbf{c}}^{(x,j)} \rangle \\ &= \langle \sum_{i\in[n]} sk_i, \mathbf{c}_k^{(x,j)} - \mathbf{t}_k^{(j)} + \hat{\mathbf{t}}_k^{(j)} \rangle,\end{aligned}$$

where by Theorem 2

$$\begin{aligned}&\langle sk_k, \mathbf{c}_k^{(x,j)} - \mathbf{t}_k^{(j)} + \hat{\mathbf{t}}_k^{(j)}\rangle = \langle sk_k, \mathbf{c}_k^{(x,j)}\rangle - \langle sk_k, \mathbf{t}_k^{(j)}\rangle + \langle sk_k, \hat{\mathbf{t}}_k^{(j)}\rangle\\ =\ & \frac{q}{p}x \cdot \mathbf{s}_k[j] + \frac{q}{p}(\mathbf{s}_k[d]-1)\cdot \mathbf{r}_k^{(j)} \cdot \mathbf{z}_k + \mathbf{e}_k\end{aligned}$$

can be directly deduced. Then consider for $i \in [n]\backslash\{k\}$

$$\begin{aligned}&\langle sk_i, \mathbf{c}_k^{(x,j)} - \mathbf{t}_k^{(j)} + \hat{\mathbf{t}}_k^{(j)}\rangle = \langle sk_i, \mathbf{c}_k^{(x,j)}\rangle - \langle sk_i, \mathbf{t}_k^{(j)}\rangle + \langle sk_i, \hat{\mathbf{t}}_k^{(j)}\rangle\\ =\ & \frac{q}{p}x \cdot \mathbf{s}_i[j] + \frac{q}{p}(\mathbf{s}_i[d]-1)\cdot \mathbf{r}_k^{(j)} \cdot \mathbf{z}_k + (\mathbf{b}_k - \mathbf{b}_i)\cdot \mathbf{r}_k^{(j)} + \mathbf{e}_i,\end{aligned}$$

holds due to

$$\sum_{l\in[m]} \mathbf{R}_k^{(j,l)}(\mathbf{g}^{-1}(\mathbf{v}[j]), 0, \cdots, 0)^\top = \sum_{l\in[m]} \hat{\mathbf{R}}_k^{(j,l)}(0, \cdots, 0, \mathbf{g}^{-1}(\mathbf{v}[j]))^\top$$

where $\mathbf{R}_k^{(j,l)} = [\mathbf{R}_k^{(j,l')}||\mathbf{R}_k^{(j,l'')}]$ and $\hat{\mathbf{R}}_k^{(j,l)} = [\mathbf{R}_k^{(j,l''')}||\mathbf{R}_k^{(j,l')}]$ as defined in $\mathsf{MKHSS.Share}$ and $\mathbf{e}_i$ is a small noise for $i \in [n]$.

Obliviously,

$$\begin{aligned}&\langle \overline{ek}_0 + \overline{ek}_1, \overline{\mathbf{c}}^{(x,j)}\rangle = \langle \sum_{i\in[n]}(ek_i^{(0)} + ek_i^{(1)}), \overline{\mathbf{c}}^{(x,j)}\rangle\\ =\ & \sum_{i\in[n]}\langle sk_i, \mathbf{c}_k^{(x,j)} - \mathbf{t}_k^{(j)} + \hat{\mathbf{t}}_k^{(j)}\rangle\\ =\ & \langle sk_k, \mathbf{c}_k^{(x,j)} - \mathbf{t}_k^{(j)} + \hat{\mathbf{t}}_k^{(j)}\rangle + \sum_{i\in[n]\backslash\{k\}}\langle sk_i, \mathbf{c}_k^{(x,j)} - \mathbf{t}_k^{(j)} + \hat{\mathbf{t}}_k^{(j)}\rangle\\ =\ & \frac{q}{p}x\cdot \mathbf{s}_k[j] + \frac{q}{p}(\mathbf{s}_k[d]-1)\cdot \mathbf{r}_k^{(j)}\cdot \mathbf{z}_k + \mathbf{e}_k\\ +\ & \sum_{i\in[n]\backslash\{k\}}(\frac{q}{p}x\cdot \mathbf{s}_i[j] + \frac{q}{p}(\mathbf{s}_i[d]-1)\cdot \mathbf{r}_k^{(j)}\cdot \mathbf{z}_k + (\mathbf{b}_k - \mathbf{b}_i)\cdot \mathbf{r}_k^{(j)} + \mathbf{e}_i)\\ =\ & \frac{q}{p}x\cdot \sum_{i\in[n]}\mathbf{s}_i[j] + \frac{q}{p}\sum_{i\in[n]}(\mathbf{s}_i[d]-1)\cdot \mathbf{r}_k^{(j)}\cdot \mathbf{z}_k + \sum_{i\in[n]\backslash\{k\}}(\mathbf{b}_k - \mathbf{b}_i)\cdot \mathbf{r}_k^{(j)} + \sum_{i\in[n]}\mathbf{e}_i\\ =\ & \frac{q}{p}x\cdot \sum_{i\in[n]}\mathbf{s}_i[j] + \sum_{i\in[n]\backslash\{k\}}(\mathbf{b}_i - \mathbf{b}_k)\cdot \mathbf{r}_k^{(j)} + \sum_{i\in[n]\backslash\{k\}}(\mathbf{b}_k - \mathbf{b}_i)\cdot \mathbf{r}_k^{(j)} + \mathbf{e}\\ =\ & \frac{q}{p}x\cdot \sum_{i\in[n]}\mathbf{s}_i[j] + \mathbf{e} = \frac{q}{p}x\cdot(\overline{ek}_0 + \overline{ek}_1)[j] + \mathbf{e}.\end{aligned}$$

That is, the extended input shares still satisfy the requirement of nearly linear decryption under the extended evaluation key in a component-wise form. For the simplicity of writing, we denote by $\langle \overline{ek}_0 + \overline{ek}_1, \overline{\mathbf{C}}^{(x)}\rangle := (q/p)\cdot x\cdot(\overline{ek}_0 + \overline{ek}_1) + \mathbf{e}$ the decryption $(\langle \overline{ek}_0 + \overline{ek}_1, \overline{\mathbf{c}}^{(x,1)}\rangle, \ldots, \langle \overline{ek}_0 + \overline{ek}_1, \overline{\mathbf{c}}^{(x,d)}\rangle) \in \mathbb{Z}_q^d$ of the matrix of d extended ciphertexts. □

Correctness of MKHSS. For any $\lambda \in \mathbb{N}$, for all inputs $x_1, \ldots, x_n \in [\mathbb{Z}]_{B_{inp}}$, for any RMS programs P with maximum number of input addition instructions P_{inp+} and the magnitude bound B_{max}, where $|P| \le poly(\lambda)$, $p/B_{max} \ge \lambda^{\omega(1)}$ and $q/(B_{max}\cdot p) \ge \lambda^{\omega(1)}$, for $(pk, sk) \leftarrow \mathsf{MKHSS.Gen}(1^\lambda)$ and $(s_i^{(0)}, s_i^{(1)}) \leftarrow \mathsf{MKHSS.Share}(i, pk, sk, x)$, there exits an PPT adversary $\mathcal{B}$ breaking the pseudo randomness of PRF with advantage holds at least

$$\begin{aligned}\mathsf{Pr}^{\mathsf{cor}}_{\mathsf{MKHSS},\{x_i\}_{i\in[n]},P,r}(\lambda) \ge 1 - \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{B}}(\lambda) - (B_{max}+1)/q\\ -|P|\cdot d\cdot P_{inp+}\cdot B_{max}\cdot(B_{ct}\cdot p/q + B_{sk}/p) - |P|\cdot d\cdot(p/q + 1/p).\end{aligned}$$

Proof. Suppose $\varepsilon_0 := \mathsf{Pr}^{\mathsf{cor}}_{\mathsf{MKHSS},\{x_i\}_{i\in[n]},P,r}(\lambda)$ denotes the probability that a program P is successfully evaluated by employing our MKHSS scheme on input $x_1,\ldots,x_n \in [\mathbb{Z}]_{B_{inp}}$. We then define $\varepsilon_1 := \mathsf{Pr}^{1}_{\mathsf{MKHSS},\{x_i\}_{i\in[n]},P,r}(\lambda)$ as the probability of correct evaluation, where we replace every evaluation of PRF with a randomly chosen value $r \leftarrow \mathbb{Z}_q^d$. And $|\varepsilon_0 - \varepsilon_1| \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{B}}(\lambda)$ has been proved by Boyle et al. in [5]. That is, if ε_0 differs significantly from ε_1, then there exits an adversary $\mathcal{B}$ who can break the pseudorandomness of the PRF used in every evaluation.

Next, a lower bound of the probability ε_1 can be easily obtained according to [5]. Specifically, if the evaluation of a program P over shares $(t_0^{(x)}, t_1^{(x)})$ corresponds to the input value $x \in \mathbb{Z}$, there is overwhelming probability over the choice of $r \leftarrow \mathbb{Z}_q^d$ that the shares satisfy $t_0^{(x)} + t_1^{(x)} = x \cdot \mathbf{s} = (x, x\cdot\hat{\mathbf{s}}) \mod q$, where $\mathbf{s} = (1,\hat{\mathbf{s}}) \in \mathbb{Z}\times\mathbb{Z}_q^{d-1}$. Further, for $x_0, x_1 \in \mathbb{Z}_q$ be random and $y \in \mathbb{Z}$, we have $x_0 + x_1 = y$ with probability at least $1 - (B_{max}+1)/q$.

Then we show that the homomorphic evaluation of P is true, i.e. $t_0^{(x)} + t_1^{(x)} = x\cdot\mathbf{s} = (x, x\cdot\hat{\mathbf{s}}) \mod q$ indeed holds. Assuming correctness of distributed decryption holds, by Lemma 3 we have

- Load: on instruction $(id, \overline{\mathbf{C}}^{(x)})$,

$$\begin{aligned}&\mathbf{t}_0^{(x)} + \mathbf{t}_1^{(x)} = \mathsf{MRegev.DDec}(0, \overline{ek}_0, \overline{\mathbf{C}}^{(x)}) + \mathsf{MRegev.DDec}(1, \overline{ek}_1, \overline{\mathbf{C}}^{(x)})\\ &= \lfloor \frac{p}{q}\cdot\langle \overline{ek}_0 + \overline{ek}_1, \overline{\mathbf{C}}^{(x)}\rangle\rceil \mod q = \lfloor \frac{p}{q}\cdot(\frac{q}{p}x\cdot(\overline{ek}_0+\overline{ek}_1)+\mathbf{e})\rceil \mod q\\ &= x\cdot(\overline{ek}_0+\overline{ek}_1) \mod q.\end{aligned}$$

- $\mathsf{Add1}$: on instruction $(id, t_b^{(x)}, t_b^{(x')})$ for $b \in \{0,1\}$,

$$\begin{aligned}&\mathbf{t}_0^{(x+x')} + \mathbf{t}_1^{(x+x')} = \mathbf{t}_0^{(x)} + \mathbf{t}_0^{(x')} + \mathbf{r} + \mathbf{t}_1^{(x)} + \mathbf{t}_1^{(x')} - \mathbf{r} \mod q\\ &= x\cdot(\overline{ek}_0+\overline{ek}_1) + x'\cdot(\overline{ek}_0+\overline{ek}_1) \mod q = (x+x')\cdot(\overline{ek}_0+\overline{ek}_1) \mod q.\end{aligned}$$

- Mult: on instruction $(id, t_b^{(x)}, \overline{\mathbf{c}}^{(x')})$ for $b \in \{0,1\}$,

$$\begin{aligned}&\mathbf{t}_0^{(x\cdot x')} + \mathbf{t}_1^{(x\cdot x')} = \mathsf{MRegev.DDec}(0, \mathbf{t}_0^{(x)}, \overline{\mathbf{C}}^{(x')})\\ &+ \mathsf{MRegev.DDec}(1, \mathbf{t}_1^{(x)}, \overline{\mathbf{C}}^{(x')}) = \lfloor \frac{p}{q}\cdot\langle \mathbf{t}_0^{(x)} + \mathbf{t}_1^{(x)}, \overline{\mathbf{C}}^{(x')}\rangle\rceil \mod q\\ &= \lfloor \frac{p}{q}\cdot(\frac{q}{p}x\cdot x'\cdot(\overline{ek}_0+\overline{ek}_1)+\mathbf{e})\rceil \mod q = x\cdot x'\cdot(\overline{ek}_0+\overline{ek}_1) \mod q.\end{aligned}$$

Here we put the addition of input values $\mathsf{Add2}$ and the output of the memory values Output out of consideration, because these two operations have no effect on the shares.

Throughout the execution of the above instructions, the intermediary values x are bounded by $\|x\|_\infty \leq B_{max}$ and the distribution of $(\mathbf{t}_0^{(x)}, \mathbf{t}_1^{(x)})$ is random subject to $\mathbf{t}_0^{(x)} + \mathbf{t}_1^{(x)} = x\cdot\mathbf{s}$. Similar to the analysis in [5], the distribute decryption fails with probability at most $P_{inp+}\cdot B_{max}\cdot(B_{ct}\cdot p/q + B_{sk}/p) + (p/q + 1/p)$.

d decryptions are required for every instruction. Consider the RMS program P with size $|P|$, a union bound over all $d\cdot|P|$ decryptions is yielded as $\varepsilon_1 \geq 1 - (B_{max}+1)/q - d\cdot|P|\cdot P_{inp+}\cdot B_{max}\cdot(B_{ct}\cdot p/q + B_{sk}/p) - d\cdot|P|\cdot(p/q+1/p)$. This completes our proof of correctness. □

Security of MKHSS. The MKHSS scheme is semantically secure.

Proof. Here we employ a hybrid argument to prove the security of our MKHSS scheme, and define 3 games in Fig. 1.

Game $\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}$ is the original MKHSS security game, corresponding to

$$\mathsf{Adv}^{\mathsf{sec}}_{\mathsf{MKHSS},\mathcal{A}}(\lambda) \leq \left|\Pr\left[\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]-1/2\right|.$$

Now suppose there exists a PPT adversary $\mathcal{A}$ which can distinguish between games $\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}$ and $\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}$, then another PPT adversary $\mathcal{B}$ that breaks the security of MRegev can be constructed as follows. On input the public key pk by the challenger of the MRegev security game and input $(b, x_0, x_1, state)$, $\mathcal{B}$ chooses $\beta \in \{0,1\}$. Then $\mathcal{B}$ draws $K \leftarrow \mathcal{K}$ and sets $ek^{(1)} = ek - ek^{(0)} \mod q$ with $ek, ek^{(0)} \xleftarrow{\$} \mathbb{Z}_q^d$. For $j \in [d]$, $\mathcal{B}$ computes $\mathbf{c}_j \leftarrow \mathsf{Regev.OKDM}(pk, x_\beta, j)$. $\mathcal{B}$ queries $\mathbf{T}^k_j \leftarrow \mathcal{O}_{\mathsf{MRegev}}(pk,*,0)$ and $\hat{\mathbf{T}}^k_j \leftarrow \mathcal{O}_{\mathsf{MRegev}}(pk,*,1)$ for all random vectors r_j used in executing $\mathsf{Regev.OKDM}$, where $j \in [d]$ and $k \in [m]$. Finally, $\mathcal{B}$ sends pk and $s^b_i = (ek^{(b)}, K, \{\mathbf{c}_j\}_{j\in[d]}, \{\mathbf{T}^k_j, \hat{\mathbf{T}}^k_j\}_{j\in[d],k\in[m]})$ to $\mathcal{A}$. $\mathcal{B}$ takes the output of $\mathcal{A}$ as its challenge response. If the MRegev oracle returns the encryption with the first component $\mathbf{P}$ of the public key pk, the distribution of s^b_i equals that in game $\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}$. While if it returns the encryption with a randomly generated public key component $\mathbf{P}'$, the distribution of s^b_i equals that in game $\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}$. Thus we have

$$\left|\Pr\left[\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]-\Pr\left[\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]\right| \leq \mathsf{Adv}^{\mathsf{ind-cpa}}_{\mathsf{MRegev},\mathcal{A}}(\lambda).$$

Considering $\mathsf{G}^2_{\mathsf{MKHSS},\mathcal{A}}$, the only difference from $\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}$ is that when the adversary queries the encryption of x_β, it returns the encryption of 0 instead of the real encryption. In this case, we can construct a PPT adversary $\mathcal{D}$ on the security of $\mathsf{Regev.OKDM}$ from a PPT adversary $\mathcal{C}$ distinguishing between $\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}$ and $\mathsf{G}^2_{\mathsf{MKHSS},\mathcal{A}}$, which is quite similar to that of $\mathcal{B}$ above. After getting pk, $(b, x_0, x_1, state)$, β, K, $ek^{(0)}$ and $ek^{(1)}$, $\mathcal{D}$ chooses random vectors r_j and computes $\mathbf{T}^k_j \leftarrow \mathsf{MRegev.Enc}(pk, r_j[k], 0)$ and $\hat{\mathbf{T}}^k_j \leftarrow \mathsf{MRegev.Enc}(pk, r_j[k], 1)$ for all r_j, where $j \in [j]$ and $k \in [m]$. In the query phase, $\mathcal{D}$ queries $\mathbf{c}_j \leftarrow \mathcal{O}_{\mathsf{KDM}}(pk, x_\beta, j)$ for $j \in [d]$. Finally, $\mathcal{D}$ sends pk and s^b_i to $\mathcal{C}$ and takes the output of $\mathcal{C}$ as its. If the KDM oracle returns the real encryption of x_β, the distribution of s^b_i equals that in game $\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}$. While if it returns the encryption of 0, the distribution of s^b_i equals that in game $\mathsf{G}^2_{\mathsf{MKHSS},\mathcal{A}}$. Therefore, It holds that

$$\left|\Pr\left[\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]-\Pr\left[\mathsf{G}^2_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]\right| \leq \mathsf{Adv}^{\mathsf{kdm-ind}}_{\mathsf{Regev.OKDM},\mathcal{A}}(\lambda).$$

Since the view of $\mathcal{C}$ in $\mathsf{G}^2_{\mathsf{MKHSS},\mathcal{A}}$ is independent of the choice on β, we have

$$\Pr\left[\mathsf{G}^2_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]=1/2.$$

According to the triangle inequality, we can put them all together and obtain that

$$\begin{aligned}&\left|\Pr\left[\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]-\Pr\left[\mathsf{G}^2_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]\right|\\ &\quad \leq \mathsf{Adv}^{\mathsf{ind-cpa}}_{\mathsf{MRegev},\mathcal{A}}(\lambda)+\mathsf{Adv}^{\mathsf{kdm-ind}}_{\mathsf{Regev.OKDM},\mathcal{A}}(\lambda).\end{aligned}$$

Further, we have given the semantical security of the KDM oracle and MRegev in Definition 2 and Theorem 1 respectively. Finally, we have

$$\left|\mathsf{Pr}\left[\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}(\lambda)=1\right]-1/2\right|\le negl(\lambda).$$

This completes our proof. □

$\underline{\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}}$:

- $(b, x_0, x_1, state) \leftarrow \mathcal{A}(1^\lambda)$
- $\beta \leftarrow \{0,1\}$
- $(pk, sk) :=$ $((\mathbf{P}, \mathbf{s}[d]), \mathbf{s}) \leftarrow \mathsf{MKHSS.Gen}(1^\lambda)$
- $(s^{(0)}, s^{(1)}) \leftarrow$ $\mathsf{MKHSS.Share}(*, pk, sk, x_\beta)$, where $\mathsf{MKHSS.Share}$ completely follows that in Sec. 4.3
- $\beta' \leftarrow \mathcal{A}(state, pk, s^{(b)})$
- if $\beta' = \beta$, return 1
- else, return 0

$\underline{\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}}$:

- $(b, x_0, x_1, state) \leftarrow \mathcal{A}(1^\lambda)$
- $\beta \leftarrow \{0,1\}$
- $(pk, sk) := ((\mathbf{P}, \mathbf{s}[d]), \mathbf{s})$ $\leftarrow \mathsf{MKHSS.Gen}(1^\lambda)$
- $(s^{(0)}, s^{(1)}) \leftarrow$ $\mathsf{MKHSS.Share}(*, pk, sk, x_\beta)$, where $\mathsf{MKHSS.Share}$ follows that in $\mathsf{G}^0_{\mathsf{MKHSS},\mathcal{A}}$, except for using $\mathbf{P}'$ (newly generated by $\mathsf{MRegev.KGen}$) instead of $\mathbf{P}$
- $\beta' \leftarrow \mathcal{A}(state, pk, s^{(b)})$
- if $\beta' = \beta$, return 1
- else, return 0

$\underline{\mathsf{G}^2_{\mathsf{MKHSS},\mathcal{A}}}$:

- $(b, x_0, x_1, state) \leftarrow \mathcal{A}(1^\lambda)$
- $\beta \leftarrow \{0,1\}$
- $(pk, sk) := ((\mathbf{P}, \mathbf{s}[d]), \mathbf{s})$ $\leftarrow \mathsf{MKHSS.Gen}(1^\lambda)$
- $(s^{(0)}, s^{(1)}) \leftarrow$ $\mathsf{MKHSS.Share}(*, pk, sk, x_\beta)$, where $\mathsf{MKHSS.Share}$ follows that in $\mathsf{G}^1_{\mathsf{MKHSS},\mathcal{A}}$, except for using $\mathsf{Regev.Enc}(\mathbf{P}, 0)$ instead of $\mathsf{Regev.OKDM}(pk, x_\beta, *)$
- $\beta' \leftarrow \mathcal{A}(state, pk, s^{(b)})$
- if $\beta' = \beta$, return 1
- else, return 0

Fig. 1. Games defined in the proof of the security of MKHSS

Context Hiding of MKHSS. The MKHSS scheme is context-hiding.

Proof. To prove the context-hiding property, we first construct the simulator $\mathcal{S}$: On input the security parameter λ and the evaluating result of the RMS program $y = P(x_1, \ldots, x_n)$ over input values $x_1, \ldots, x_n \in [\mathbb{Z}]_{B_{inp}}$, $\mathcal{S}$ chooses $y'_0 \leftarrow \mathbb{Z}_q$, and let $y'_1 = y - y'_0 \mod q$. Finally, the simulator outputs (y'_0, y'_1), which is a random distribution in $\mathbb{Z}_q$.

Consider the output (y_0, y_1) of $\mathsf{MKHSS.Eval}$ is a random distribution in $\mathbb{Z}_q$, it is straightforward to know that (y_0, y_1) is statistically indistinguishable from (y'_0, y'_1). □

Lemma 4. *Assuming hardness of* $\mathsf{LWE}_{\mathsf{q,d-1,m},\chi_{\mathsf{sk}},\chi}$*, the noise bound of the above* $(n, 2)$*-MKHSS scheme is* $B_{ct} = (2m^2nl + 1) \cdot B_{err}$.

Proof. For the extension ciphertext denoted by $\overline{\mathbf{c}}^{(x,j)} := \mathbf{c}_k^{(x,j)} - \mathbf{t}_k^{(j)} + \hat{\mathbf{t}}_k^{(j)}$, we get the new noise term $\mathbf{e}_k + \sum_{i\in[n]\setminus\{k\}} \mathbf{e}_i = \mathbf{e} + \sum_{j=1}^m \mathbf{eR}(\mathbf{g}^{-1}(\mathbf{z}[j]), 0, \cdots, 0)^\top + \sum_{j=1}^m \mathbf{eR}(0, \cdots, 0, \mathbf{g}^{-1}(\mathbf{z}[j]))^\top$ of $\langle \overline{\mathbf{sk}}, \overline{\mathbf{c}}^{(x,j)} \rangle$ according to Lemma 3. Since $\|\mathbf{e}\|_\infty \le B_{err}$, the noise bound B_{ct} can be easily concluded by the maximum of the above noise term, that is $(2m^2nl + 1) \cdot B_{err}$. □

5 Performance Analysis

In this section, we first give the theoretical analysis of our MKHSS scheme from both the client-side and the server-side costs. Then, we implement our scheme and evaluate its performance by conducting comprehensive experiments.

5.1 Theoretical Analysis

We give an overview of communication and storage costs in Table 1, along with Table 2 for an overview of computation costs on the server-side, and compare them with the single-key HSS scheme proposed in [5].

On the Client-Side. To support multi-key evaluation, each client generates two more sets of MRegev ciphertexts than those in [5] and adds them to input shares. In that output shares returned by servers remain the same sizes, there is no additional computational cost added to the output client in terms of reconstruction. On the client-side, MKHSS trades some computation and communication efficiency in share generation phase for richer functionality.

On the Server-Side. In the evaluation phase, the size of the extended shares remains the same due to the use of homomorphic linear combination. Thus, the evaluation performed by each server over the extended shares is completely identical to that in [5]. On the server-side, the costs of storage and communication of MKHSS remain the same as those of [5], and the extra computational overhead caused by share extension is tolerable because of the powerful computational abilities of servers.

Comparison with [12]. The HSS scheme in [12] can support the functionality similar to ours by using multi-key HE as its underlying building block. In the two-server setting, to compute a k-degree polynomial, the computation complexity of our MKHSS scheme is $O(k \cdot d^2)$ on the server-side, as oppose to $O(2^{2^{\frac{k-1}{2}}})$ for the construction of [12] in the fair case. To be specific, there are 2^{k-1} k-degree terms to be calculated by each server for [12]. Because of costly key-switching operations required by the underlying $\frac{k-1}{2}$-degree HE scheme, evaluation of high degree polynomials in [12] is apparently expensive. Furthermore, high-degree HE schemes suffer from ciphertext expansion more or less. That is, the size of an extended ciphertext at least increases linearly with the number of input clients, which causes the time spent by the output client for decryption to reconstruct the final result becomes larger. In contrast, our MKHSS scheme has fixed-size extended ciphertexts, avoiding costly multi-key HE, and only a simple modular addition is needed for reconstruction. Obviously, our MKHSS scheme gives it more advantages in terms of high-degree polynomial evaluations. We only give a theoretical comparison with [12], since no LWE-based multi-key homomorphic encryption scheme compatible with its construction can be used for instantiation.

Table 1. Sizes of Input Shares/Extended Ciphertexts/Memory Values/Output Shares

	input shares	(extended) ciphertexts	memory values	output shares
[5]	$\|K\| + d + d^2$	d^2	d	d
ours	$\|K\| + d + (4lm + 1)d^2$	d^2	d	d

$|K|$ denotes PRF key size and the units of values in the table are $\mathbb{Z}_q$.

Table 2. Dominant computation costs of computing servers

	ciphertext extension	load	add1(mem)	add2(input)	mult(mem − input)
[5]	–	$d^2 \cdot mul_q$	$d \cdot add_q$	$d^2 \cdot add_q$	$d^2 \cdot mul_q$
ours	$n \cdot dml \cdot mul_q$	$d^2 \cdot mul_q$	$d \cdot add_q$	$d^2 \cdot add_q$	$d^2 \cdot mul_q$

mul_q and add_q respectively denote the number of multiplication and addition over $\mathbb{Z}_q$.

5.2 Experimental Results

Refer to the parameter selection method in [5], we start with choosing the plaintext bound B_{max} and set $r = B_{max}$ to maximum the output space. Also, we choose a statistical security parameter $\kappa = 40$, the number of *Add2* instructions $P_{inp+} = 1$, the noise distribution $B_{err} = 8\sigma$ with $\sigma = 8$, and the secret key bound $B_{sk} = 1$. We adapt the correctness of MKHSS to ensure each multiplication has a failure probability no more than $2^{-\kappa}$, which means $d \cdot B_{max} \cdot (B_{ct} \cdot p/q + B_{sk}/p) \leq 2^{-\kappa}$ should be required. We set $p = d \cdot B_{max} \cdot B_{sk} \cdot 2^{\kappa}$, then a bound on q is given as $q \geq 2^{\kappa+1} \cdot m^2 \cdot p \cdot d \cdot B_{max} \cdot B_{err} \cdot B_{sk}$ by combining the formula from Lemma 4.

Our experiments are conducted over a Ubuntu 20.0.4 LTS 64-bit operating system with Intel Xeon Gold 6230R 2.1 GHz×26 processors and 256 GB of RAM. All mathematical calculations involving large integers were realized based on the C++ libraries GMP-6.2.1 and NTL-11.5.1. And the library cymric is used as the secure random number generator in our public-key setting. Table 3 shows the running time of 5 RMS instructions in MKHSS.Eval algorithm corresponding to the different parameters. To the best of our knowledge, this is the first implementation of a multi-key HSS scheme from LWE. Note that in order to make the evaluation results plainer, we did not leverage any speedup tricks, such as parallelism, in the implementation, which would significantly improve the performance. For example, we can get about a 5× improvement when multithreading (the number of threads is 5) is applied, making the latency for low degree polynomial tolerable.

Table 3. The running time (in milliseconds) of 5 RMS instructions in MKHSS.Eval

B_{max}	d	q	p	security	load	add1	add2	mult	output
2	2^{12}	2^{137}	2^{53}	$2^{103.3}$	2618.03	0.483	1848.54	2592.62	4×10^{-3}
2^{16}	2^{12}	2^{167}	2^{68}	$2^{83.74}$	2724.66	0.504	2037.37	2708.18	7×10^{-3}
2^{32}	2^{13}	2^{203}	2^{85}	$2^{142.0}$	11202.55	0.943	7734.02	11191.72	6×10^{-3}
2^{64}	2^{13}	2^{267}	2^{117}	$2^{104.9}$	10756.25	0.942	7667.27	10694.5	4×10^{-3}
2^{128}	2^{14}	2^{399}	2^{182}	$2^{143.9}$	63199.07	1.922	31386.75	55646.32	5×10^{-3}
2^{256}	2^{14}	2^{655}	2^{310}	$2^{84.6}$	70796.85	1.937	34906.75	69374.8	7×10^{-3}

6 Conclusions

In this paper, we mainly focus on the construction of homomorphic secret sharing in the multi-key setting. We first propose a formal definition of multi-key HSS in the two-server model. Then by applying the homomorphic linear combination for the modified Regev encryption to the single-key construction in [5], we present an instantiation of multi-key HSS scheme based on LWE without multi-key HE. It allows independent input clients to share private data between two non-colluding servers, so that polynomials interpreted as RMS programs can be jointly computed over these input shares. In particular, the number of input clients and the degree of evaluated polynomials can be as high as a polynomial in the security parameter. Besides, the security and context hiding property of our scheme preserve the privacy of input clients against the computing servers and the output client. In the future, we will explore the HSS construction in the multi-server model with the threshold malicious assumption, and further reduce the computation and communication costs of our MKHSS to meet the practical requirements better.

References

1. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_29
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 1–10 (1988)
3. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_19
4. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: optimizing rounds, communication, and computation. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 163–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_6
5. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 3–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_1
6. Catalano, D., Fiore, D.: Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1518–1529 (2015)
7. Chen, X., Zhang, L.F.: Two-server verifiable homomorphic secret sharing for high-degree polynomials. In: Susilo, W., Deng, R.H., Guo, F., Li, Y., Intan, R. (eds.) ISC 2020. LNCS, vol. 12472, pp. 75–91. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62974-8_5

8. Chor, B., Gilboa, N.: Computationally private information retrieval (extended abstract). In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 304–313 (1997)
9. Cleve, R.: Towards optimal simulations of formulas by bounded-width programs. Comput. Complex. **1**, 91–105 (1991)
10. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_9
11. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_4
12. Eriguchi, R., Nuida, K.: Homomorphic secret sharing for multipartite and general adversary structures supporting parallel evaluation of low-degree polynomials. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13091, pp. 191–221. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92075-3_7
13. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith, W.E.: Homomorphic secret sharing from Paillier encryption. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) ProvSec 2017. LNCS, vol. 10592, pp. 381–399. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68637-0_23
14. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theor **31**(4), 469–472 (1985)
15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, pp. 169–178 (2009)
16. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
17. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, pp. 218–229 (1987)
18. Jiang, B.: Multi-key FHE without ciphertext-expansion in two-server model. Front. Comput. Sci. **16**(1), 1–8 (2022). https://doi.org/10.1007/s11704-021-0479-5
19. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_3
20. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
21. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_26
22. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 84–93 (2005)
23. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Found. Secure Comput. **4**(11), 169–179 (1978)

24. Tsaloli, G., Liang, B., Mitrokotsa, A.: Verifiable homomorphic secret sharing. In: Baek, J., Susilo, W., Kim, J. (eds.) ProvSec 2018. LNCS, vol. 11192, pp. 40–55. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01446-9_3
25. Tsaloli, G., Mitrokotsa, A.: Sum it up: verifiable additive homomorphic secret sharing. In: Seo, J.H. (ed.) ICISC 2019. LNCS, vol. 11975, pp. 115–132. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40921-0_7

Identity-Based Encryption from Lattices Using Approximate Trapdoors

Malika Izabachène[1], Lucas Prabel[2](✉), and Adeline Roux-Langlois[3]

[1] Independent Scholar, Paris, France
[2] Univ Rennes, CNRS, IRISA, Rennes, France
lucas.prabel@irisa.fr
[3] Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000, Caen, France

Abstract. Practical implementations of advanced lattice-based constructions have received much attention since the first practical scheme instantiated over NTRU lattices, proposed by Prest et al. (Asiacrypt 2014). They are using powerful lattice-based building blocks which allow to build Gaussian preimage sampling and trapdoor generation efficiently. In this paper, we propose two different constructions and implementations of identity-based encryption schemes (IBE) using approximate variants of "gadget-based" trapdoors introduced by Chen et al. (Asiacrypt 2019). Both constructions are proven secure.

Our first IBE scheme is an adaptation of the Bert et al. scheme (PQCrypto 2021) to the approximate setting, relying on the Module-LWE hardness assumption and making use of the Micciancio-Peikert paradigm with approximate trapdoors. The second IBE relies on a variant of the NTRU hardness assumption.

We provide several timings and a comparison analysis to explain our results. The two different instantiations give interesting trade-offs in terms of security and efficiency and both benefit from the use of approximate trapdoors. Though our second IBE construction is less efficient than other NTRU-based IBEs, we believe our work provides useful insights into efficient advanced lattice-based constructions.

Keywords: Lattice-based cryptography · approximate trapdoors · Gaussian preimage sampling · module lattices · IBE

1 Introduction

Identity-based encryption (IBE) is an advanced public key encryption scheme in which an identity, such as a username, email address, or social security number, acts as the public key. In identity-based encryption, the sender encrypts a message with the unique identity of the recipient, and the recipient then decrypts

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-35486-1_13.

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 270–290, 2023.
https://doi.org/10.1007/978-3-031-35486-1_13

the ciphertext with their private key to obtain the original message. In this way, one party can send an encrypted message to any other party without requesting the recipient's public key beforehand. The pair "identity" and "associated secret key" acts as a classical public key and secret key pair in a classical public key encryption scheme.

The idea was to eliminate the need for a public certificate across email systems. These schemes allowed secure communication without exchanging user keys. In [31] Shamir presented a solution for an identity-based signature scheme but the first IBE constructions appeared only in 2001 in [6,10] and were based respectively on bilinear maps and quadratic residue-based assumptions. However, these schemes were vulnerable to quantum attacks due to Shor's algorithm.

Lattice-Based IBE Constructions. In [18], Gentry, Peikert and Vaikuntanathan described the first lattice-based IBE, relying on the Dual-Regev encryption scheme. An important contribution of their work was a sampling algorithm (known as GPV sampling) which showed how to use a short basis as a trapdoor for generating short lattice vectors. This sampler was then used to construct a lattice-based IBE scheme, proven adaptively secure against chosen-plaintext attack in the random oracle model as defined in [6,10]. However, the master public key and user secret keys had large sizes in $O(n^2)$ bits. Later on, a construction of a Hierarchical IBE (HIBE) scheme in the standard model was proposed in [8] based on a new mechanism for users' keys delegation. This IBE scheme was proven secure in the selective model where the adversary needs to target an identity beforehand. In 2010, Agrawal et al. [1] proposed a Learning With Errors (LWE)-based IBE scheme with a trapdoor structure and with performance comparable to the GPV scheme. Their construction viewed an identity as a sequence of bits and then assigned a matrix to each bit. It used a sampling algorithm to obtain a basis with low Gram-Schmidt norm for the master secret key and formed a lattice family with two associated trapdoors to generate short vectors; one for all lattices in the family and the other one for all but one.

The first Ring-LWE based IBE scheme has been proposed by Ducas, Lyubashevsky and Prest [11] (DLP-IBE), which is still considered the most efficient scheme to date due to smaller key sizes. The use of the ring variant increased efficiency by reducing the public key size and ciphertext size to $O(n)$. The security of their scheme holds in the random oracle model and is related to the NTRU hardness assumption. An efficient C implementation of the DLP-IBE scheme and a detailed performance analysis was provided in [27]. In 2017, Campbell and Grover introduced a HIBE scheme, called LATTE, which can be viewed as a combination of the DLP scheme with the delegation mechanism from [8]. An optimized implementation and refined analysis of LATTE, has recently been proposed in [33].

The work from [5] constructed an IBE using the notion of gadget-based trapdoors in the ring setting, introduced by [28]. Such trapdoors can be seen as linear transformations mapping hard instances of cryptographic problems on some lattices to easy instances on a lattice defined by a public "gadget matrix". The IBE from [5] also made use of the efficient Gaussian preimage sampling algorithms from [17] to propose an implementation of their scheme. In [4], this IBE and its associated sampling algorithms were adapted to the module setting and instantiated. The use of module lattices of dimension nd, where d is the rank module, led to a more flexible choice of parameters. In [32], the authors proposed new efficient gadget sampling algorithms which didn't need floating-point arithmetic, and as fast as the original [17] sampler.

All those constructions generally make use of dedicated trapdoors, needed by the authority to generate the secret key of a user. In that case, building the trapdoor and sampling particular short vectors are quite costly, and represent the main bottleneck in the efficiency of such schemes. More recently, [9] introduced approximate trapdoors to improve the efficiency of schemes built from lattice trapdoors while keeping the same concrete security. [16] showed that those approximate trapdoors, relying on the [28] framework, exist on a family of NTRU lattices. Our work explores the application of those approximate trapdoors, and in particular of this family of NTRU lattices, to more advanced schemes where the generation of private keys in a multi-user setting is needed.

Our Contributions. Our main contribution is to provide and implement two lattice-based IBE schemes (IBE1 and IBE2), which make use of families of gadget-based approximate trapdoors. Our two constructions rely on the LWE problem over modules (Module-LWE) and the inhomogeneous NTRU problem (iNTRU) respectively. We investigate how to instantiate and parametrize the approximate trapdoor preimage sampling over these two families of trapdoors in a way to obtain provable and efficient quantum-safe IBE schemes. The IBE1 construction is an adaptation of the identity-based encryption scheme from [1,5] to the module setting, using approximate trapdoors. The IBE2 construction follows the same blueprint as the DLP scheme except that it makes use of iNTRU gadget-based approximate trapdoors. As in previous IBE constructions, encryption is based on the Dual-Regev encryption scheme. We provide a complete public and open-source C implementation[1] with performance benchmarking. The implementation is modular and makes it easy to change the building blocks of our algorithms according to the desired properties that we want to get.

Our work explores the use of approximate trapdoors for the construction of IBE schemes and the potential practical insights we can gain from it. We first adapted the construction of [4] using approximate trapdoors in our IBE1. The error induced by the approximate setting requires changes at several levels, either for the choice of encoding or for the sampling algorithms. As expected, we obtain better timings for all four algorithms composing our IBE by using approximate trapdoors rather than exact ones.

The second scheme IBE2 makes use of approximate trapdoors relying on a variant of NTRU rather than Module-LWE. Our approach was motivated by the fact that other efficient IBEs, such as DLP and Latte, used the NTRU hardness assumption. However, unlike us, these last two schemes used the GPV paradigm to generate trapdoors, which significantly changes the way their schemes are constructed compared to ours.

[1] https://github.com/lucasprabel/approx_lattice.

More Details on Our Implementation Choices. Our IBE1 proof relies on a statistical trapdoor instantiation. Although the size of the parameters increases consequently, the use of approximate trapdoors allowed us to mitigate this loss in efficiency induced by the use of a statistical instantiation. In order to ensure decryption correctness, we also need to use a large modulus (see Sect. 4.1). This leads us to perform calculations carefully on 64-bit integers, so as not to affect our scheme efficiency. The IBE1 also makes use of a small-norm encoding, instead of the low-degree encoding used in [4] to ensure that the noise is still not too large in order to decrypt. The encoding we use sets constraints on the structure of the ring $\mathcal{R}_q$ which is not compatible with the NTT for polynomial multiplications. Instead, we use a "partial NTT" based on [26] results, which reduces multiplication in $\mathcal{R}_q$ to multiplication in smaller rings. We also have optimized the underlying CRT representation algorithm compared to [4]. Finally, the sampling algorithms have been adapted to the approximate setting. Table 2 shows some applicable parameter sets together with their concrete bit security using the LWE estimator from [2] with BKZ as a reduction cost model. All the algorithms comprising the IBE1 over module lattices are more efficient than their exact counterpart at the same security level. This improvement concerns in particular Setup and Extract which are optimized by a factor ≈1.5. We give more details in Sect. 4.2.

The IBE2 scheme is instantiated using gadget-based trapdoors on a family of NTRU lattices. Table 4 provides a set of applicable parameter sets together with their concrete bit security. The computational trapdoors instantiation lowers the bounds on parameters required for the correctness compared to the IBE1 scheme, which allows the use of smaller moduli. For the IBE2 construction, an identity is encoded as $\mathsf{H}(\mathsf{id})$ with H having special properties so that we are able to respond to private key queries except for the target identity chosen by the adversary.

Table 1. Timings comparison in ms of the different operations of the IBE2 scheme in this paper and in Latte [33] for different sets of parameters.

Scheme	$(n, \lceil log_2(q) \rceil)$	Security level	Setup	Extract	Encrypt	Decrypt
[33]	$(1024, 24)$	128	102	0.82	0.05	0.06
[33]	$(2048, 25)$	256	292	2.62	0.10	0.13
This paper	$(1024, 25)$	159	3.32	5.92	1.10	0.07
This paper	$(2048, 25)$	293	10.21	12.79	2.96	0.16

In Table 1, we provide timings comparison with the [33] IBE scheme. A more complete and detailed comparison is given in Sect. 5.2 between our IBE2 scheme and the [11] and [33] IBE schemes which also rely on the NTRU assumption. The use of approximate trapdoors and of the iNTRU assumption allows to obtain better timings for the Setup algorithm and for the Extract algorithm for some sets of parameters. Unfortunately, we obtain an overhead for encryption which essentially comes from the Gaussian sampling phase. We can save a factor 3.5 using binomial samples as in [11] and [33] but there is still an overhead due to the extra number of samples we need. We make $n(2+m)$ calls to the integer Gaussian sampler while encryption in [11] and [33] makes use of $3n$ binomial sampling calls, where n is the dimension of the underlying polynomial ring and m is the size of one of the vector used in encryption. As an example, we obtain a timing of 0.87ms for one encryption against 0.13 for [33] for a security level of 128 bits.

Organization of the Paper. The paper is structured as follows: Sect. 2 reviews the necessary linear algebra and lattice backgrounds. The Section also presents the notions of approximate trapdoor. Then, Sect. 4 and Sect. 5 contain a detailed description of the two IBEs, based on the $\mathsf{Module\text{-}LWE}$ hardness assumption and based on a variant of the NTRU assumption respectively. Both Sections provide a security analysis and detailed performance results together with a comparison of their respective analogue in terms of lattice-based building blocks and assumptions.

2 Preliminaries

Notations. Throughout the paper, the vectors are written in lowercase bold letters (e.g. $\boldsymbol{v}$) and matrices in uppercase bold letters (e.g. $\boldsymbol{A}$). We refer to their entries with a subscript index v_i, $A_{i,j}$. We denote $\|\cdot\|$ and $\|\cdot\|_\infty$ the euclidean norm and the infinity norm respectively. The norm of a vector over $\mathbb{Z}_q$ is the norm of the corresponding vector over $\mathbb{Z}$ obtained by choosing its entries in the set $\{-\lfloor q/2 \rfloor, \dots, \lfloor q/2 \rfloor\}$. The norm of a polynomial $a = \sum_{i=0}^{n-1} a_i X^i$ is the norm of the coefficients vector $(a_0, \dots, a_{n-1})$. Finally, the norm of a matrix is the maximum norm of its column vectors. If x is sampled from a distribution D, we write $x \leftarrow D$. We denote $\mathcal{U}(S)$ the uniform distribution over a finite set S.

We say that a function f is negligible when $f(n) = o(n^{-c})$ for all $c > 0$ as $n \to \infty$. An event is said to happen with overwhelming probability if its probability of not happening is negligible. Two distributions D_0 and D_1 over the same countable domain Ω are said to be statistically indistinguishable if their statistical distance $\Delta(D_0, D_1) = \frac{1}{2}\sum_{\omega \in \Omega} |D_0(\omega) - D_1(\omega)|$ is negligible. Two distributions are said to be computationally indistinguishable if no probabilistic polynomial time algorithm can distinguish them with non-negligible advantage.

2.1 Lattices Background and Discrete Gaussian Distributions

Lattices. A lattice Λ in $\mathbb{R}^m$ is the set $\left\{\sum_{i=1}^{k} \lambda_i \boldsymbol{b}_i, \lambda_i \in \mathbb{Z}\right\}$ of all integer linear combinations of some linearly independent vectors $\boldsymbol{B} = \{\boldsymbol{b}_1, \dots, \boldsymbol{b}_k\} \subset \mathbb{R}^m$. We call k the rank of the lattice and m its dimension. When $k = m$, the lattice is said to be full-rank.

Given $\boldsymbol{A} \in \mathbb{Z}_q^{n \times m}$ and $\boldsymbol{u} \in \mathbb{Z}_q^n$, we define the following m-dimensional q-ary lattice $\Lambda_q^\perp(\boldsymbol{A}) = \{\boldsymbol{x} \in \mathbb{Z}^m | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \bmod q\}$, and its coset $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}) = \{\boldsymbol{x} \in \mathbb{Z}^m | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{u} \bmod q\}$.

Module lattices are particular lattices that have a polynomial structure. We denote d the module rank of those lattices. When $d = 1$, module lattices are in fact ideal lattices. We consider the ones that are based on the rings $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ and $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle X^n + 1\rangle$, where n is a power of two and q is prime. They are sublattices of the full lattice $\mathcal{R}^m$, which is isomorphic to the integer lattice $\mathbb{Z}^{nm}$.

Gaussian Distributions. We recall that a symmetric matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is positive definite (resp. positive semidefinite) if $\boldsymbol{x}^T\boldsymbol{M}\boldsymbol{x} > 0$ (resp. $\boldsymbol{x}^T\boldsymbol{M}\boldsymbol{x} \geq 0$) for all nonzero $\boldsymbol{x} \in \mathbb{R}^n$. In this case we write $\boldsymbol{M} \succ 0$ (resp. $\boldsymbol{M} \succeq 0$). We say that $\boldsymbol{M} \succeq \boldsymbol{N}$ when $\boldsymbol{M} - \boldsymbol{N} \succeq 0$, and write $\boldsymbol{M} \succeq \eta$ instead of $\boldsymbol{M} \succeq \eta\boldsymbol{I}_n$ when $\eta \geq 0$ is a real. The spherical continuous Gaussian function of center $\boldsymbol{c} \in \mathbb{R}^n$ and parameter σ is defined on $\mathbb{R}^n$ by $\rho_{\boldsymbol{c},\sigma}(\boldsymbol{x}) = \exp(-\frac{\pi\|\boldsymbol{x}-\boldsymbol{c}\|^2}{\sigma^2})$. For a positive definite matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$, we also define the (skewed) Gaussian function $\rho_{c,\sqrt{\boldsymbol{\Sigma}}}(\boldsymbol{x}) = \exp(-\pi(\boldsymbol{x}-\boldsymbol{c})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{c}))$. Then, for a full-rank lattice $\Lambda \subset \mathbb{Z}^n$, we denote $D_{\Lambda,\sigma,\boldsymbol{c}}$ (respectively $D_{\Lambda,\sqrt{\boldsymbol{\Sigma}},\boldsymbol{c}}$) the spherical (resp. ellipsoid) discrete Gaussian distribution of center $\boldsymbol{c} \in \mathbb{R}^n$ and parameter $\sigma > 0$, associated with the density $\rho_{\boldsymbol{c},\sigma}$ (resp. $\rho_{c,\sqrt{\boldsymbol{\Sigma}}}$).

Smoothing Parameter. The smoothing parameter of a lattice Λ, denoted $\eta_\varepsilon(\Lambda)$, was first introduced in [29]. Lemma 1 gives an upper bound on it.

Lemma 1 ([18, Lemma 3.1]). *Let $\Lambda \subset \mathbb{R}^n$ be a lattice with basis $\boldsymbol{B}$, and $\tilde{\boldsymbol{B}}$ the Gram-Schmidt orthogonalization of $\boldsymbol{B}$. Then, for any $\varepsilon > 0$, we have $\eta_\varepsilon(\Lambda) \leq \|*\|\tilde{\boldsymbol{B}} \cdot \sqrt{\ln(2n(1+1/\varepsilon))/\pi}$.*

Gaussian Tailcut. We will use the following result to bound the euclidean norm of vectors that follow a discrete Gaussian distribution.

Lemma 2 ([24, Lemma 4.4]). *For any integer $n \geq 1$ and reals $\sigma > 0$, $t > 1$, $\Pr\left[\|\boldsymbol{x}\| > t\sigma\sqrt{n} \mid \boldsymbol{x} \leftarrow D_{\mathbb{Z}^n,\sigma}\right] < t^n e^{\frac{n}{2}(1-t^2)}$.*

We call t a tailcut of the discrete Gaussian of parameter σ when a vector $\boldsymbol{x}$ sampled from $D_{\mathbb{Z}^n,\sigma}$ verifies the inequality $\|\boldsymbol{x}\| \leq t\sigma\sqrt{n}$ on its euclidean norm with overwhelming probability.

2.2 Cryptographic Problems on Lattices

We work on the rings $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ and $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle X^n + 1\rangle$, where n is a power of two and q a prime modulus. Let us now define the hard problems on which our constructions rely. In most practical lattice-based schemes, security is based on structured variants of LWE rather than on LWE itself. In the Module-SIS and Module-LWE variants, the parameter d is the rank of the module, and nd is the dimension of the corresponding module lattice. The hardness of those problems is proven by worst-case to average-case reductions from hard problems on module lattices (see [22]).

Definition 1 ($\mathsf{Module\text{-}SIS}_{n,d,m,q,\beta}$)**.** *Given a uniform $\boldsymbol{A} \in \mathcal{R}_q^{d\times m}$, find a vector $\boldsymbol{x} \in \mathcal{R}^m$ such that $\boldsymbol{Ax} = \boldsymbol{0} \mod q$, and $0 < \|\boldsymbol{x}\| \leq \beta$.*

Definition 2 ($\mathsf{Decision\ Module\text{-}LWE}_{n,d,q,\sigma}$)**.** *Given a uniform $\boldsymbol{A} \in \mathcal{R}_q^{d\times m}$ and the vector $\boldsymbol{b}^T = \boldsymbol{s}^T\boldsymbol{A} + \boldsymbol{e}^T \bmod q \in \mathcal{R}_q^m$, where $\boldsymbol{s} \leftarrow \mathcal{U}(\mathcal{R}_q^d)$ and $\boldsymbol{e} \leftarrow D_{\mathcal{R}^m,\sigma}$, distinguish the distribution of $(\boldsymbol{A}, \boldsymbol{b})$ from the uniform distribution over $\mathcal{R}_q^{d\times m} \times \mathcal{R}_q^m$.*

As stated in [7], the $\mathsf{Module\text{-}LWE}$ problem remains hard when the short secret $\boldsymbol{s}$ is sampled from the Gaussian distribution $D_{\mathcal{R}^d,\sigma}$.

Note that $\mathsf{Module\text{-}LWE}_{n,d,q,\sigma}$ with rank $d = 1$ corresponds to the ring version of LWE, which will be denoted $\mathsf{Ring\text{-}LWE}_{n,q,\sigma}$. We define $\boldsymbol{G} = \boldsymbol{I}_d \otimes \boldsymbol{g}^T$ and its associated lattice named the module $\boldsymbol{G}$-lattice, for which the $\mathsf{Module\text{-}SIS}$ problem is easy.

To build our second IBE, we use gadget-based trapdoors whose pseudorandomness is based on the inhomogeneous NTRU problem (iNTRU), a variant of NTRU introduced in [15].

Definition 3 ($\mathsf{iNTRU}_{q,\chi}$)**.** *Let k, q be integers and χ a distribution over $\mathcal{R}$. The input of the $\mathsf{iNTRU}_{q,\chi}$ problem is a vector $\boldsymbol{a} \in \mathcal{R}_q^k$ which is either taken uniform in $\mathcal{R}_q^k$ or either set as $\boldsymbol{a} = r^{-1}(\boldsymbol{g} + \boldsymbol{e})$ where $(r, \boldsymbol{e})$ is drawn from χ^{k+1}. The goal is to decide which is the case.*

In their paper, the authors from [15] showed a reduction of a matrix-variant of iNTRU called MiNTRU from the non-standard $\mathsf{Ring\text{-}LWE}$ with a trapdoor oracle access problem (more details are given in [15, Section 4.3]). The reduction can be adapted to the iNTRU problem as well.

2.3 Encoding Identities with Full-Rank Differences

We use the notion of full-rank differences encoding (FRD) in our first scheme, with different properties that the ones used in [1].

Definition 4 ([4])**.** *An* encoding with full-rank differences *from the set $\mathcal{M}$ to a ring $\mathcal{R}$ is a map $H : \mathcal{M} \longrightarrow \mathcal{R}$ such that:*

- *for any $m \in \mathcal{M}$, $H(m)$ is invertible,*
- *for any $m_1, m_2 \in \mathcal{M}$ such that $m_1 \neq m_2$, $H(m_1) - H(m_2)$ is invertible,*
- *H is computable in polynomial time.*

As shown in [4], we construct an FRD encoding in the module setting (i.e. over $\mathcal{R}_q^{d\times d}$), by first constructing one in the ring setting (i.e. over $\mathcal{R}_q$). The encodings used in our IBE scheme impose a structure on the ring $\mathcal{R}_q$ which is not compatible with the Number Theoretic Transform (NTT). To speed up polynomial multiplications and to mitigate the loss of performance due to the inability of using the NTT, we use ideas from [26]. This method can be thought of as a "partial NTT". It is based on the following result.

Lemma 3 **([26, Corollary 1.2]).** *Let $n \geq r > 1$ be powers of 2, and q a prime such that $q \equiv 2r + 1 \pmod{4r}$. Then the cyclotomic polynomial $X^n + 1$ factors in $\mathbb{Z}_q[X]$ as $X^n + 1 = \prod_{i=1}^{r} \left(X^{n/r} - s_i\right)$, for some distinct $s_i \in \mathbb{Z}_q^*$ such that the $\left(X^{n/r} - s_i\right)$ are irreducible in $\mathbb{Z}_q[X]$. Moreover, if $y \in \mathbb{Z}_q[X]/(X^n + 1)$ satisfies $0 < \|y\|_\infty < \frac{q^{1/r}}{\sqrt{r}}$, then y has an inverse in $\mathbb{Z}_q[X]/(X^n + 1)$.*

This lemma was used in [4] to build a "low-degree" FRD (the identities were encoded as polynomials of low-degree). In our case, we construct a "small-norm" FRD, encoding polynomials with ℓ_∞-norm smaller than $\frac{q^{1/r}}{2\sqrt{r}}$. We make use of the "small-norm" FRD described in Proposition 1 rather than "low-degree" FRD encoding so that the correctness of the decryption algorithm still holds when using approximate trapdoors, because they introduce an additional error term that must be bounded.

Proposition 1. *Let $n \geq r > 1$ be powers of 2, q a prime such that $q \equiv 2r + 1 \pmod{4r}$ and $1 \leq D \leq \frac{q^{1/r}}{2\sqrt{r}}$ an integer. We define $\mathcal{M} = \{-D, \ldots, D\}^n \setminus \{(0, \ldots, 0)\}$ the set of identities. Then the following map $H_M : \mathcal{M} \longrightarrow \mathcal{R}_q$, such that $H_M(m_0, \ldots, m_{n-1}) = \sum_{i=0}^{n-1} m_i X^i$ is an FRD encoding.*

Proof. We have $\|H_M(m)\|_\infty \leq D < \frac{q^{1/r}}{2\sqrt{r}}$ for all m because of the choice of $\mathcal{M}$. So according to Lemma 3, $H_M(m)$ is invertible. For all $m_1, m_2 \in \mathcal{M}$, we also have $\|H_M(m_1) - H_M(m_2)\|_\infty \leq 2D < \frac{q^{1/r}}{\sqrt{r}}$ so $H_M(m_1) - H_M(m_2)$ is invertible. Finally, H_M is an FRD encoding. □

FRD on Modules. As explained in [4], FRD encoding in the module setting can be built using an existing FRD encoding in the ring setting $H_M : \mathcal{M} \longrightarrow \mathcal{R}_q$ by constructing $H_M(m) \cdot \boldsymbol{I}_d \in R_q^{d \times d}$ for $m \in \mathcal{M}$ and $\boldsymbol{I}_d \in \mathcal{R}_q^{d \times d}$ is the identity matrix.

3 Trapdoors on Lattices

The two IBE constructions make use of efficient trapdoors, called gadget-trapdoors, introduced in [28]. Such trapdoors were generalized to ideal lattices in [21] and to module lattices in [4]. An efficient instantiation of the sampling algorithms was given in [17]. All those results allow us to efficiently instantiate trapdoor generation and sampling algorithms. Having introduced the $\boldsymbol{G}$-lattice in the previous Section, we can now define the associated $\boldsymbol{G}$-trapdoors.

Definition 5. *A $\boldsymbol{G}$-trapdoor for a matrix $\boldsymbol{A} \in \mathcal{R}_q^{d \times m}$ is a matrix $\boldsymbol{R} \in \mathcal{R}^{(m-dk) \times dk}$ such that*

$$\boldsymbol{A} \begin{bmatrix} \boldsymbol{R} \\ \hline \boldsymbol{I}_{dk} \end{bmatrix} = \boldsymbol{H}\boldsymbol{G}$$

for some invertible $\boldsymbol{H} \in \mathcal{R}_q^{d \times d}$, called the tag *of $\boldsymbol{A}$.*

The knowledge of the trapdoor $\boldsymbol{R}$ allows one to compute small vectors on any coset of a given lattice, by sampling on the $\boldsymbol{G}$-lattice first, and then by using $\boldsymbol{R}$ to map the sample to a small vector in the given lattice.

3.1 Approximate Trapdoors for Module Lattices

As shown in [9], the gadget trapdoor proposed by Micciancio and Peikert can be modified to an approximate trapdoor, in a way that further reduces the sizes of the public matrix, the trapdoor and the preimage.

Definition 6 ($\mathsf{AISIS}_{n,d,m,q,\alpha,\beta}$)**.** *For any $n, d, m, q \in \mathbb{N}$ and $\alpha, \beta \in \mathbb{R}$, the approximate inhomogeneous short integer solution problem $\mathsf{AISIS}_{n,d,m,q,\alpha,\beta}$ is defined as follows: given $\boldsymbol{A} \in \mathcal{R}_q^{d\times m}, \boldsymbol{y} \in \mathcal{R}_q^d$, find a vector $\boldsymbol{x} \in \mathcal{R}^m$ such that $||\boldsymbol{x}|| \leq \beta$ and such that there is a vector $\boldsymbol{z} \in \mathcal{R}^d$ satisfying: $||\boldsymbol{z}|| \leq \alpha$ and $\boldsymbol{Ax} = \boldsymbol{y} + \boldsymbol{z}$ mod q.*

Definition 7 (Approximate Trapdoor). *A string τ is called an (α, β)-approxi mate trapdoor for a matrix $\boldsymbol{A} \in \mathcal{R}_q^{d\times m}$ if there is a probabilistic polynomial time algorithm that given $\tau, \boldsymbol{A}$ and any $\boldsymbol{y} \in \mathcal{R}_q^d$, outputs a non-zero vector $\boldsymbol{x} \in \mathcal{R}^m$ such that $||\boldsymbol{x}|| \leq \beta$ and there is a vector $\boldsymbol{z} \in \mathcal{R}^d$ satisfying $||\boldsymbol{z}|| \leq \alpha$ and $\boldsymbol{Ax} = \boldsymbol{y} + \boldsymbol{z}$ mod q.*

In practice, we generate approximate trapdoors by dropping ℓ entries corresponding to the small powers of b from the gadget matrix $\boldsymbol{G}$ to get the following matrix: $\boldsymbol{F} = \boldsymbol{I}_d \otimes \boldsymbol{f}^T = \boldsymbol{I}_d \otimes \left[b^\ell \ b^{\ell+1} \ b^{\ell+2} \cdots b^{k-1}\right] \in \mathcal{R}^{d\times d(k-\ell)}$.

3.2 Module-LWE Approximate Trapdoors

In [9], the authors described the approximate trapdoor generation and the approximate preimage sampling algorithms, adapted from [28] by making use of the approximate gadget matrix $\boldsymbol{F}$ instead of $\boldsymbol{G}$. Its trapdoor generation and trapdoor preimage sampling algorithms over modules are recalled in Fig. 1 and the result of the approximate sampling is stated in Theorem 1.

Theorem 1 ([9, Theorem 4.1])**.** *There exists probabilistic, polynomial time algorithms* ApproxTrapGen1(·) *and* ApproxSamplePre1(·) *such that:*

1. ApproxTrapGen1$(\boldsymbol{H}, \sigma)$ *takes as input public parameters, a tag matrix $\boldsymbol{H} \in \mathcal{R}^{d\times d}$ and parameter $\sigma > 0$ and returns a matrix-approximate trapdoor pair $(\boldsymbol{A}, \boldsymbol{R}) \in \mathcal{R}_q^{d\times m} \times \mathcal{R}^{\bar{m}\times\omega}$ where $\boldsymbol{R}$ coefficients are drawn from a Gaussian distribution of parameter σ over $\mathcal{R}$.*
2. *Let $\boldsymbol{A}$ be generated with an approximate trapdoor as above. The following two distributions are statistically indistinguishable:*

$$\{(\boldsymbol{A}, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{y}) \mid \boldsymbol{u} \leftarrow \mathcal{U}(\mathcal{R}_q^d), \boldsymbol{x} \leftarrow \text{ApproxSamplePre1}(\boldsymbol{A}, \boldsymbol{R}, \boldsymbol{0}, \boldsymbol{u}, \zeta), \boldsymbol{y} = \boldsymbol{u} - \boldsymbol{Ax} \ mod \ q\}$$
$$and \ \{(\boldsymbol{A}, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{y}) \mid \boldsymbol{x} \leftarrow D_{\mathcal{R}^m,\zeta}, \boldsymbol{y} \leftarrow D_{\mathcal{R}^d,\sigma\sqrt{(b^{2\ell}-1)/(b^2-1)}}, \boldsymbol{u} = \boldsymbol{Ax} + \boldsymbol{y} \ mod \ q\}$$

for any $\sigma \geq \sqrt{b^2+1} \cdot \omega(\sqrt{\log d})$ and $\zeta \gtrsim \sqrt{b^2+1}\frac{s_1^2(\boldsymbol{R})}{s_{2d}(\boldsymbol{R})}\eta_\epsilon(\mathbb{Z}^{dk})$.

3.3 iNTRU Approximate Trapdoors

In [16], Genise and Li introduced a family of Ring-SIS approximate trapdoors whose pseudorandomness is based on the iNTRU problem and then showed that the efficient gadget-based trapdoor framework of [28] exists on a family of NTRU lattices. Their trapdoor scheme enjoys small secret keys and is compatible with applications requiring tag matrices.

In their second trapdoor scheme, the matrix $\begin{bmatrix} -\boldsymbol{e}^T \\ r\boldsymbol{I} \end{bmatrix}$ is used as a $\boldsymbol{f}$-trapdoor for $\begin{bmatrix} 1 \ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} 1 \ r^{-1}(\boldsymbol{f}^T + \boldsymbol{e}^T) \end{bmatrix}$ where $(r, \boldsymbol{e}) \longleftarrow \chi^{k-\ell+1}$ is drawn from a distribution with small entries and $\boldsymbol{f}$ is the approximate gadget vector. They got the following results, similar to the ones from [9].

```
ApproxTrapGen1($\boldsymbol{H}, \sigma$)
1. $\bar{\boldsymbol{A}} \leftarrow \mathcal{U}(\mathcal{R}_q^{d\times\bar{m}})$
2. $\boldsymbol{R} \leftarrow D_{\mathcal{R}^{\bar{m}\times\omega},\sigma}$
3. set $\boldsymbol{A} = [\,\bar{\boldsymbol{A}} \,\big|\, \boldsymbol{HF} - \bar{\boldsymbol{A}}\boldsymbol{R}\,] \in \mathcal{R}_q^{d\times m}$
4. return $(\boldsymbol{A}, \boldsymbol{R})$

ApproxSamplePre1($\boldsymbol{A}, \boldsymbol{R}, \boldsymbol{H}, \boldsymbol{u}, \zeta$)
1. sample perturbation $\boldsymbol{p} \leftarrow D_{\mathcal{R}^m,\sqrt{\boldsymbol{\Sigma_p}}}$
2. set coset $\boldsymbol{v} = \boldsymbol{H}^{-1}(\boldsymbol{u} - \boldsymbol{Ap})$
3. sample $\boldsymbol{z} = (\boldsymbol{z_1}^T, \boldsymbol{z_2}^T)^T \leftarrow D_{\Lambda_q^{\boldsymbol{v}}(\boldsymbol{G}),\sigma_g}$
4. set $\boldsymbol{x} = \boldsymbol{p} + \left[\begin{smallmatrix}\boldsymbol{R}\\ \boldsymbol{I}\end{smallmatrix}\right]\boldsymbol{z_2} \in \mathcal{R}^m$
5. return $\boldsymbol{x}$
```

```
ApproxTrapGen2($\sigma$)
1. sample $r \longleftarrow \mathcal{D}_{\mathcal{R},\sigma}$, $\boldsymbol{e} \longleftarrow \mathcal{D}_{\mathcal{R}^m,\sigma}$
2. set $\boldsymbol{a}' = r^{-1}(\boldsymbol{f} + \boldsymbol{e}) \in \mathcal{R}^m$
3. set $\boldsymbol{a} = (1, \boldsymbol{a}') \in \mathcal{R}_q^{m+1}$
4. set $\boldsymbol{R} = \begin{bmatrix} -\boldsymbol{e}^T \\ r\boldsymbol{I}_m \end{bmatrix} \in \mathcal{R}^{(m+1)\times m}$
5. return $(\boldsymbol{a}, \boldsymbol{R})$

ApproxSamplePre2($\boldsymbol{a}, \boldsymbol{R}, u, \zeta$)
1. sample perturbation $\boldsymbol{p} \leftarrow D_{\mathcal{R}^{m+1},\sqrt{\boldsymbol{\Sigma_p}}}$
2. set coset $v = (u - \boldsymbol{a}^T\boldsymbol{p}) \in \mathcal{R}_q$
3. sample $\boldsymbol{z} = (\boldsymbol{z_1}^T, \boldsymbol{z_2}^T)^T \leftarrow D_{\Lambda_q^v(\boldsymbol{g}^T),\sigma_g}$
4. set $\boldsymbol{x} = \boldsymbol{p} + \boldsymbol{R}\boldsymbol{z_2} \in \mathcal{R}^{m+1}$
5. return $\boldsymbol{x}$
```

Fig. 1. Description of the approximate trapdoor generation and preimage sampling algorithms based on [9] (on the left-hand side) and [16] (on the right-hand side) instantiations, where the perturbation is sampled with parameter $\boldsymbol{\Sigma_p} = \zeta^2\boldsymbol{I}_m - \sigma_g^2\left[\begin{smallmatrix}\boldsymbol{R}\\ \boldsymbol{I}\end{smallmatrix}\right]\left[\begin{smallmatrix}\boldsymbol{R}^T & \boldsymbol{I}\end{smallmatrix}\right]$ on the left side and $\boldsymbol{\Sigma_p} = \zeta^2\boldsymbol{I}_{m+1} - \sigma_g^2\boldsymbol{R}\boldsymbol{R}^T$ on the right side. Both are independent of the target $\boldsymbol{u}$. We refer to [9,16] for more information on the iNTRU based trapdoor generation.

Theorem 2. *Let $r \leftarrow \chi$ and $\boldsymbol{e}^T \leftarrow \chi^m$ and set the trapdoor function description as $\boldsymbol{a} = \begin{bmatrix} 1 \ \boldsymbol{a}' \end{bmatrix} = \begin{bmatrix} 1 \ r^{-1}(\boldsymbol{f} + \boldsymbol{e}) \end{bmatrix} \in \mathcal{R}_q^{m+1}$. Let $\eta = \eta_\epsilon(\mathbb{Z}^{n\times m})$ and $\sigma_g = \eta_\epsilon(\Lambda_q^\perp(\boldsymbol{g}^T)) \geq \sqrt{b^2+1} \cdot \eta_\epsilon(\mathbb{Z}^{n\times m})$ for some $\epsilon \in (0,1)$ and $\zeta \succeq \sqrt{\sigma_g^2\boldsymbol{R}\boldsymbol{R}^T + \eta^2\boldsymbol{I}_{m+1}}$. Then, the following distributions are within a max-log distance $3\log\frac{1+\epsilon}{1-\epsilon} \leq \frac{6\epsilon}{1-\epsilon}$:*

$$\{(\boldsymbol{a}, \boldsymbol{x}, u, y) \mid u \leftarrow \mathcal{U}(\mathcal{R}_q), \boldsymbol{x} \leftarrow \text{ApproxSamplePre2}(\boldsymbol{a}, \boldsymbol{R}, u, \zeta), y = u - \boldsymbol{a}^T\boldsymbol{x} \in \mathcal{R}_q\}$$
$$\textit{and } \{(\boldsymbol{a}, \boldsymbol{x}, u, y) \mid \boldsymbol{x} \leftarrow D_{\mathcal{R}^{m+1},\zeta}, y \leftarrow D_{\mathcal{R},\sigma_e} \mod q, u = \boldsymbol{a}^T\boldsymbol{x} + y \in \mathcal{R}_q\}$$

for $\sigma_e = \sigma_g\sqrt{(b^{2\ell}-1)/(b^2-1)}$.

4 An Approximate-IBE Scheme on Modules

The first IBE scheme we present is the IBE scheme from [1,5] adapted to the module setting, instantiated using the approximate trapdoors from [9]. At a high level, the scheme will make use of the following blocks:

- The master secret key is an $\boldsymbol{F}$-approximate trapdoor $\boldsymbol{R} \in \mathcal{R}^{\bar{m}\times\omega}$ associated with $\boldsymbol{A}$ with tag $\mathbf{0}$, with subgaussian coefficients of parameter σ, with $\omega = d(k-\ell)$. The master public key is a tuple consisting of a uniformly random vector $\boldsymbol{u} \in \mathcal{R}_q^d$ and the matrix $\boldsymbol{A} \in \mathcal{R}_q^{d\times m}$, with $m = \bar{m} + \omega$ chosen as: $\boldsymbol{A} = \left[\bar{\boldsymbol{A}} \,\middle|\, -\bar{\boldsymbol{A}}\boldsymbol{R}\right]$. Taking $\bar{m} = d\log q$, we get that $\bar{\boldsymbol{A}}$ is full rank with high probability according to [7, Lemma 2.6]. Moreover, by taking $\sigma > 4\cdot 4^{\frac{1}{nd}}\sqrt{n}$, we obtain that $\boldsymbol{A}$ is statistically close to uniform by Corollary 7.5 from [25].
- A "small-norm" FRD encoding H_M as described in Sect. 2.3. This allows anyone, with the knowledge of the master public key $\boldsymbol{A}$, to compute a public matrix $\boldsymbol{A}_{\mathsf{id}} = \boldsymbol{A} + \left[\mathbf{0}_{d,\bar{m}} \,\middle|\, \boldsymbol{H}_{\mathsf{id}}\boldsymbol{F}\right]$ associated with the identity id of a user. Then, the secret key for id is an approximate short vector $\boldsymbol{x}_{\mathsf{id}} \in \mathcal{R}^m$ obtained by using the ApproxSamplePre1 algorithm. Such a vector satisfies the relation $\boldsymbol{A}_{\mathsf{id}}\boldsymbol{x}_{\mathsf{id}} \approx \boldsymbol{u}\ mod\ q$. We can bound the approximate error in this relation by using Theorem 1 and the fact that we use a small-norm FRD encoding.
- Finally, we use the Dual-Regev encryption scheme for the encryption and decryption algorithms, taking care of the additional error which appears when using approximate trapdoors.

4.1 Construction

We detail the first IBE where users' keys are defined based on the approximate preimage sampling from Theorem 1 and encryption is based on the Dual-Regev scheme.

- Setup$(1^n) \longrightarrow (\mathsf{mpk}, \mathsf{msk})$:
  - $(\boldsymbol{A}, \boldsymbol{R}) \leftarrow \text{ApproxTrapGen1}(\mathbf{0}, \sigma) \in \mathcal{R}_q^{d\times m} \times \mathcal{R}^{\bar{m}\times w}$, $\boldsymbol{u} \leftarrow \mathcal{U}(\mathcal{R}_q^d)$;
  - $\mathsf{mpk} = (\boldsymbol{A}, \boldsymbol{u})$, $\mathsf{msk} = \boldsymbol{R}$.
- Extract$(\mathsf{mpk}, \mathsf{msk}, \mathsf{id}) \longrightarrow \mathsf{sk}_{\mathsf{id}}\ \boldsymbol{x} \in \mathcal{R}^m$:
  - $\boldsymbol{H}_{\mathsf{id}} \leftarrow H_M(\mathsf{id})$; $\boldsymbol{A}_{\mathsf{id}} \leftarrow \boldsymbol{A} + \left[\mathbf{0}_{d,\bar{m}} \,\middle|\, \boldsymbol{H}_{\mathsf{id}}\boldsymbol{F}\right] \in \mathcal{R}_q^{d\times m}$;
  - $\boldsymbol{x} \leftarrow \text{ApproxSamplePre1}(\boldsymbol{A}_{\mathsf{id}}, \boldsymbol{R}, \boldsymbol{H}_{\mathsf{id}}, \boldsymbol{u}, \zeta)$;
- Encrypt$(\mathsf{mpk}, \mathsf{id}, M) \longrightarrow C = (\boldsymbol{b}, c) \in \mathcal{R}_q^{m+1}$:
  - $\boldsymbol{H}_{\mathsf{id}} \leftarrow H_M(\mathsf{id})$; $\boldsymbol{A}_{\mathsf{id}} \leftarrow \boldsymbol{A} + \left[\mathbf{0}_{d,\bar{m}} \,\middle|\, \boldsymbol{H}_{\mathsf{id}}\boldsymbol{F}\right] \in \mathcal{R}_q^{d\times m}$;
  - $\boldsymbol{s} \leftarrow D_{\mathcal{R}_q^d,\tau}$, $\boldsymbol{e}_{\mathbf{0}} \leftarrow D_{\mathcal{R}^{m-w},\tau}$, $\boldsymbol{e}_{\mathbf{1}} \leftarrow D_{\mathcal{R}^{w},\gamma}$, and $e' \longleftarrow D_{\mathcal{R},\tau}$;
  - $\boldsymbol{b} \leftarrow (\boldsymbol{s}^T\boldsymbol{A}_{\mathsf{id}})^T + (\boldsymbol{e}_{\mathbf{0}}^T \mid \boldsymbol{e}_{\mathbf{1}}^T)^T$ and $c \leftarrow \boldsymbol{s}^T\boldsymbol{u} + e' + \lfloor q/2 \rfloor M$;
- Decrypt$(\mathsf{sk}_{\mathsf{id}}, C) \rightarrow M$:
  - set $\boldsymbol{x} = \mathsf{sk}_{\mathsf{id}}$ and compute $\mathsf{res} \leftarrow c - \boldsymbol{b}^T\boldsymbol{x}$ which has integer coefficients;
  - for each i, if the coefficient $\mathsf{res}_i \in \mathbb{Z}$ is closer to $\lfloor q/2 \rfloor$ than to 0, then $M_i = 1$, otherwise $M_i = 0$.

Correctness. To use approximate trapdoors with the Dual-Regev approach, we need to sample the LWE secret term with a small norm instead of sampling from the uniform distribution, in order to maintain the correctness of the schemes. Let's write $\boldsymbol{y} \in \mathcal{R}_q^d$ the additional error we get by using approximate trapdoors instead of exact ones. The correctness of the decryption holds if the error term $\|e' - (\boldsymbol{e_0}^T \mid \boldsymbol{e_1}^T)\boldsymbol{x} - \boldsymbol{s}^T\boldsymbol{y}\|$ is small enough, i.e. less than $\lfloor q/4 \rfloor$.

$$\begin{aligned}
\mathsf{res} &= c - \boldsymbol{b}^T\boldsymbol{x} \\
&= \boldsymbol{u}\cdot\boldsymbol{s} + e' + \lfloor q/2 \rfloor M - \left[(\boldsymbol{s}^T\boldsymbol{A}_{id})^T + (\boldsymbol{e_0}^T \mid \boldsymbol{e_1}^T)^T\right]^T \boldsymbol{x} \\
&= \boldsymbol{u}\cdot\boldsymbol{s} + e' + \lfloor q/2 \rfloor M - \boldsymbol{s}^T\boldsymbol{A}_{id}\boldsymbol{x} - (\boldsymbol{e_0}^T \mid \boldsymbol{e_1}^T)\boldsymbol{x} \\
&= \boldsymbol{u}\cdot\boldsymbol{s} + e' + \lfloor q/2 \rfloor M - \boldsymbol{s}^T(\boldsymbol{u}+\boldsymbol{y}) - (\boldsymbol{e_0}^T \mid \boldsymbol{e_1}^T)\boldsymbol{x} \\
&= \underbrace{e' - (\boldsymbol{e_0}^T \mid \boldsymbol{e_1}^T)\boldsymbol{x} - \boldsymbol{s}^T\boldsymbol{y}}_{\text{error term}} + \lfloor q/2 \rfloor M \in \mathcal{R}.
\end{aligned}$$

So we need to choose our parameters properly for the correctness of the Dual-Regev encryption to hold. We can bound as follows the euclidean norms of the quantities that appear in the error term:

- $\|e'\| \le t\tau\sqrt{n}$ from Lemma 2.
- $\|\boldsymbol{e_0}^T\boldsymbol{x_0}\| \le 2t^2\tau\zeta nd$ from Lemma 2 and Theorem 1.
- $\|\boldsymbol{e_1}^T\boldsymbol{x_1}\| \le t^2\gamma\zeta nd(k-\ell)$ from Lemma 2 and Theorem 1.
- $\|\boldsymbol{s}^T\boldsymbol{y}\| \le t^2\tau n^{5/2} d\sigma_g \frac{q^{1/r}}{\sqrt{r}}\sqrt{(b^{2\ell}-1)/(b^2-1)}$ from Lemma 2 and Theorem 1.

By substituting these bounds, we get the following constraints:

$$\begin{aligned}
\|e' - (\boldsymbol{e_0}^T \mid \boldsymbol{e_1}^T)\boldsymbol{x} - \boldsymbol{s}^T\boldsymbol{y}\| &\le \|e'\| + \|(\boldsymbol{e_0}^T \mid \boldsymbol{e_1}^T)\| + \|\boldsymbol{s}^T\boldsymbol{y}\| \\
&\le t\tau\sqrt{n} + t^2 nd\left[\zeta\left(2\tau + \gamma(k-\ell)\right) + \tau n^{3/2}\sigma_g \frac{q^{1/r}}{\sqrt{r}}\sqrt{(b^{2\ell}-1)/(b^2-1)}\right] \\
&\le \lfloor q/4 \rfloor .
\end{aligned}$$

Parameter Constraints. The following constraints, combined with the norms bounds above, must be met to ensure correctness:

- The Gaussian parameter σ_g used for the $\boldsymbol{G}$-sampling in the ApproxSamplePre1 algorithm must verify $\sigma_g \ge \sqrt{2b}\cdot(2b+1)\cdot\sqrt{\log(2nw(1+1/\epsilon))/\pi}$ (see [17], Corollary 3.1).
- The Gaussian width for preimage sampling ζ must follow the condition $\zeta > \sqrt{(\sigma_g^2+1)s_1^2(\boldsymbol{R}) + \eta_\varepsilon^2(\mathbb{Z}^{nm})}$, knowing that $s_1(\boldsymbol{R}) < 1.1\sigma(\sqrt{2nd}+\sqrt{nw}+4.7)$ with high probability (see [4], Section A.3), where $s_1(\boldsymbol{R})$ is the spectral norm of the trapdoor $\boldsymbol{R}$.
- The Gaussian width for approximate trapdoor generation σ must verify $\sigma > 4\cdot 4^{\frac{1}{nd}}\sqrt{n}$ to ensure the public matrix $\boldsymbol{A}$ we use is statistically close to uniform (see [25], Corollary 7.5).

- We choose to set the Gaussian parameter γ of the Gaussian error $\boldsymbol{e}_1 \in \mathcal{R}^w$ as $\gamma^2 = \sigma^2\|\boldsymbol{e}_0\|^2 + 2nt^2\sigma^2\tau^2$.

The proof of the following Theorem 3 is standard and can be found in the Supplementary materials, Appendix B.

Theorem 3. *The IBE construction with parameters $n, d, m, q, k, \ell, \sigma, \alpha, \zeta, \tau$ and γ, chosen as in the above description, is IND-sID-CPA secure in the standard model under the hardness of* $\mathsf{Module\text{-}LWE}_{n,d,q,\tau}$.

4.2 Implementation and Performance

The scheme is implemented in C, using [4] libraries, inheriting its modularity. It relies on several basic blocks that can be swapped out: the arithmetic over $\mathbb{Z}_q$ and $\mathcal{R}_q$, a pseudorandom number generator, and a (constant-time) sampler of discrete Gaussian distributions over $\mathbb{Z}$. To generate our specific discrete Gaussian distributions, we make use of the following building blocks: an AES-based pseudorandom number generator (implemented using AES-NI instructions for x86 architectures), and a sampler of discrete Gaussians over $\mathbb{Z}$ similar to Karney's sampler [19]. We chose this sampler as it can generate samples in constant time, independently of the center, Gaussian parameter, and output value. All the computations that deal with non-integers are carried out with floating-point operations that do not involve subnormal numbers. We rely on results from [26, Lemma 3] to reduce multiplications in $\mathcal{R}_q$ to polynomials multiplications in rings of the form $\mathbb{Z}_q[X]/\langle X^n+1\rangle$. The CRT reduction we used then allowed us to speed up polynomial arithmetic in $\mathcal{R}_q$.

To assess the security of each parameter set, we estimate the pseudorandomness of the public key (corresponding to the LWE security) and the hardness of breaking AISIS. The estimation of the LWE security is done with the LWE estimator of [2] with BKZ as the reduction model. We approximate our instances by an instance of an unstructured LWE problem in dimension nd. We follow a very pessimistic core-SVP hardness, where the cost of a BKZ algorithm with blocksize κ is taken to be the cost of only one call to an SVP oracle in dimension κ. For the AISIS problem, we follow the approach of [9] which consists in computing the smallest blocksize achieving the target root Hermite factor corresponding to forging a signature.

Timings. As expected, the 4 algorithms are more efficient for low value of d. Concerning the Decrypt algorithm, its execution time relies mostly on the value of n rather than d. Our timings have been obtained on an Intel i7-8650U CPU running at 1.9 GHz, and then scaled at 4.2GHz to compare ourselves with other schemes. We provide concrete parameters sets and the associated concrete results in Table 2.

Table 2. Proposed IBE parameters for our first construction of Sect. 4 with different pairs of polynomial ring dimension n and rank d, for different moduli sizes and taking $\sigma_g = 54.9$, $\sigma = 64.1$. σ_g is the Gaussian parameter for the $\boldsymbol{G}$-Sampling, σ is the Gaussian width of the trapdoor $\boldsymbol{R}$ used in the Setup algorithm and ζ is the standard deviation for the Gaussian preimage sampling in the Extract algorithm. Timings in columns Setup-to-Encrypt are given in ms. $\mathsf{M-LWE}_{n,d,q,\tau}$ denotes the concrete bit security of the scheme.

$\lceil\log_2 q\rceil$	(n, d)	ℓ	ζ	Setup	Extract	Encrypt	Decrypt	$\mathsf{M-LWE}_{n,d,q,\tau}$
58	$(256, 4)$	0	1137729	203.52	56.88	17.69	2.72	81
58	$(256, 4)$	8	1068989	174.94	49.82	15.00	2.40	80
58	$(256, 4)$	15	1004226	152.70	41.57	12.79	2.19	78
60	$(512, 3)$	0	1398812	210.85	67.28	19.20	4.06	110
60	$(512, 3)$	8	1315427	189.35	59.33	17.13	3.80	109
60	$(512, 3)$	15	1236981	160.33	53.39	14.00	3.22	107

Table 3. Comparison of timings in ms of the different operations of the IBE scheme between this paper and [4] for parameters giving an equivalent level of security.

Scheme	(n, d)	Setup	Extract	Encrypt	Decrypt
[4] ($\ell = 0$)	$(512, 2)$	123.62	45.80	13.12	2.91
[4] ($\ell = 0$)	$(256, 4)$	234.45	67.90	17.88	2.69
This paper ($\ell = 15$)	$(512, 2)$	79.10	29.44	10.09	2.36
This paper ($\ell = 15$)	$(256, 4)$	152.70	41.57	12.79	2.19

Comparison with Related Work. We compare our IBE performance with the IBE from [4], which corresponds to the case $\ell = 0$, that is to say the use of exact trapdoors instead of approximate ones. We observe that for a fixed pair (n, d), the larger ℓ is, the better timings are. Overall, the use of approximate trapdoors allows us to obtain better timings for all the algorithms comprising the IBE scheme.

5 An IBE Scheme Based on the Hardness of iNTRU

In this Section, we introduce the IBE2 scheme, instantiated using gadget-based approximate trapdoors over iNTRU trapdoors combined with the Dual-Regev encryption scheme over modules.

We recall that $m = k - \ell$ where $k = \lceil\log_b q\rceil$ and that the approximate gadget vector $\boldsymbol{f}$ is defined as $\boldsymbol{f}^T = \left[b^\ell \; b^{\ell+1} \; b^{\ell+2} \cdots b^{k-1}\right] \in \mathcal{R}^{k-\ell}$. Here, the master public key is a vector $\boldsymbol{a} \in \mathcal{R}_q^{m+1}$ generated with the ApproxTrapGen2 algorithm and whose pseudorandomness is based on the iNTRU problem. The master secret key $r, \boldsymbol{e} \in \mathcal{R}^{m+1}$ defines an $\boldsymbol{f}$-approximate trapdoor associated with $\boldsymbol{a}$. An identity is mapped to an element in $\mathcal{R}_q$ by the use of a hash function modeled as a random oracle in the security proof; the secret key associated with an identity id is an approximate short vector $\boldsymbol{x} \in \mathcal{R}^{m+1}$.

5.1 IBE Construction Details

We detail below the four algorithms of the IBE2:

- Setup$(1^n) \longrightarrow (\mathsf{mpk}, \mathsf{msk})$:
  - let $\boldsymbol{a} = \begin{bmatrix} 1 \ \boldsymbol{a_0}^T \end{bmatrix}^T \in \mathcal{R}_q^{m+1}$ and $\boldsymbol{R} = \begin{bmatrix} -\boldsymbol{e}^T \\ r\boldsymbol{I}_m \end{bmatrix} \in \mathcal{R}^{(m+1)\times m}$ output by ApproxTrapGen2(σ); and let $\mathcal{H} : \{0,1\}^\star \to \mathcal{R}_q$ a hash function;
  - output $\mathsf{mpk} = (\boldsymbol{a}, \mathcal{H})$ and $\mathsf{msk} = \boldsymbol{R}$.
- Extract$(\mathsf{mpk}, \mathsf{msk}, \mathsf{id}) \to \boldsymbol{x}_{\mathsf{id}} = \boldsymbol{x}_2 \in \mathcal{R}^m$:
  - define the tag $h_{\mathsf{id}} = \mathcal{H}(\mathsf{id}) \in \mathcal{R}_q$;
  - sample a short preimage $\boldsymbol{x} = (x_1, \boldsymbol{x_2}^T)^T \leftarrow$ ApproxSamplePre2$(\boldsymbol{a}, \boldsymbol{R}, h_{\mathsf{id}}, \zeta)$;
- Encrypt$(\mathsf{mpk}, \mathsf{id}, M) \to \boldsymbol{C} = (\boldsymbol{b}, c) \in \mathcal{R}_q^{m+1}$:
  - compute $h_{\mathsf{id}} = \mathcal{H}(\mathsf{id})$; sample $s \leftarrow D_{\mathcal{R},\tau}$, $\boldsymbol{e_1} \leftarrow D_{\mathcal{R}^m,\tau}, e_2 \leftarrow D_{\mathcal{R},\tau}$;
  - compute $\boldsymbol{b} = s\boldsymbol{a}_0 + \boldsymbol{e}_1 \in \mathcal{R}_q^m$ and $c = h_{\mathsf{id}} \cdot s + e_2 + \lfloor q/2 \rfloor M \in \mathcal{R}_q$, where a message is encoded as $M \in \mathcal{R}_2$;
- Decrypt$(\boldsymbol{x}_{\mathsf{id}}, \boldsymbol{C}) \to M$:
  - parse $\boldsymbol{x}_{\mathsf{id}}$ as $(x_1, \boldsymbol{x}_2)$; and compute $\mathsf{res} \leftarrow c - \boldsymbol{b}^T\boldsymbol{x}_2$;
  - for each i, if the coefficient $\mathsf{res}_i \in \mathbb{Z}$ is closer to $\lfloor q/2 \rfloor$ than to 0, $M_i = 1$, otherwise $M_i = 0$.

Correctness. We have the following equality:

$$\begin{aligned} c - \boldsymbol{b}^T\boldsymbol{x}_2 &= h_{\mathsf{id}} \cdot s + e_2 + \lfloor q/2 \rfloor M - (s\boldsymbol{a}_0 + \boldsymbol{e_1})^T\boldsymbol{x_2} \\ &= s \cdot x_1 + s\boldsymbol{a_0}^T\boldsymbol{x_2} - y \cdot s + e_2 + \lfloor q/2 \rfloor M - s\boldsymbol{a_0}^T\boldsymbol{x_2} - \boldsymbol{e_1}^T\boldsymbol{x_2} \\ &= \lfloor q/2 \rfloor M + s \cdot (x_1 - y) + e_2 - \boldsymbol{e_1}^T\boldsymbol{x_2}. \end{aligned}$$

Furthermore, the following bounds apply:

- $\|s \cdot x_1\| \le t^2\tau\zeta n$ from Lemma 2 and Theorem 2.
- $\|y \cdot s\| \le t^2\tau\sigma_g\sqrt{(b^{2\ell}-1)/(b^2-1)}n$ from Lemma 2 and Theorem 2.
- $\|e_2\| \le t\tau\sqrt{n}$ from Lemma 2.
- $\|\boldsymbol{e_1}^T\boldsymbol{x_2}\| \le t^2\tau\zeta nm$ from Lemma 2 and Theorem 2.

By substituting these bounds, we obtain:

$$\begin{aligned} \|s \cdot (x_1 - y) + e_2 - \boldsymbol{e_1}^T\boldsymbol{x_2}\| &\le \|s \cdot x_1\| + \|y \cdot s\| + \|e_2\| + \|\boldsymbol{e_1}^T\boldsymbol{x_2}\| \\ &\le t^2\tau\left[\zeta\,(m+1) + \sigma_g\sqrt{(b^{2\ell}-1)/(b^2-1)}\right] + t\tau\sqrt{n} \\ &\le \lfloor q/4 \rfloor. \end{aligned}$$

Parameters Constraints. The following constraints combined with the errors norm constraints above should be satisfied:

- The Gaussian parameter σ_g used for the $\boldsymbol{G}$-sampling in the ApproxSamplePre2 algorithm must verify $\sigma_g \geq \sqrt{2b} \cdot (2b+1) \cdot \sqrt{\log(2nw(1+1/\epsilon))/\pi}$ (see [17], Corollary 3.1).
- The Gaussian width for preimage sampling ζ must follow the condition $\zeta > \sqrt{(\sigma_g^2+1)s_1^2(\boldsymbol{R}) + \eta_\varepsilon^2(\mathbb{Z}^{nm})}$, where $s_1(\boldsymbol{R})$ is the spectral norm of the trapdoor $\boldsymbol{R}$ (see [4], Section A.3).

The proof of the following is adapted from the GPV IBE proof [18, Section 7.2] and is given in the Supplementary materials, Appendix C.

Theorem 4. *Our IBE construction with parameters $n, m, q, k, \ell, \sigma, \sigma_g, \zeta, \tau$ and γ is IND-sID-CPA secure in the random oracle model under the hardness of* $\mathsf{iNTRU}_{q,D_{R,\sigma}}$ *and* $\mathsf{Ring\text{-}LWE}_{n,q,\tau}$.

5.2 Implementation and Performance

In [15], the authors only propose a reduction from a non-standard version of LWE to iNTRU. Therefore, in the absence of a thorough study on the asymptotic and practical security of the iNTRU problem, which we leave for future work, we have chosen to estimate the security of our iNTRU instances by relying on the existing cryptanalysis on NTRU. As explained in [15], there is a syntactic link between NTRU and iNTRU. To the best of our knowledge, there is no known reduction between the two problems and the analysis of the iNTRU assumption might deserve additional study. Still, we additionally consider the practical security of NTRU for our targetted sets of parameters and we take into account the known cryptanalysis efforts on iNTRU.

For security estimation, we follow the same approach as in [16] to assess the concrete security of the IBE2 scheme. We determine the hardness of our underlying lattice problem by computing the root Hermite factor, introduced in [14]. Then, we use the following heuristic relation between the blocksize κ and the root Hermite factor δ to find the smallest blocksize which would break our underlying lattice problem:

$$\delta \approx (\frac{\kappa}{2\pi e}(\pi\kappa)^{1/\kappa})^{1/2(\kappa-1)}.$$

Finally, our experiments estimate the running time of the BKZ algorithm to analyze the concrete security of the scheme. This algorithm makes use of an oracle to solve the Shortest Vector Problem (SVP) in smaller lattices. We chose the "Core-SVP" model introduced in [3] in the sieving regime as the SVP oracle for the BKZ algorithm with time complexity $2^{0.292\kappa+16.4}$ in the blocksize κ.

Overstretched Parameters. [23] adapts the attack from [20] on iNTRU, which applies to the parameters originally proposed for the homomorphic encryption scheme from [15]. The attack can be performed when the modulus q is much larger than the dimension of the associated lattices. The target iNTRU instance has an error and secret that follow a uniform binary distribution. Following [20] attack, [23] uses the fact that a very dense sublattice can be found in this NTRU-like lattice, because of the overstretched regime. Relying on a lemma from Pataki and Tural [30, Lemma 1], they can bound the volume of this sublattice and run a BKZ-reduction which leads to a full recovery of the iNTRU secret. [12] improves the asymptotic bound given by Kirchner and Fouque [20] by conducting a refined analysis which lowers the overstretched regime for NTRU with ternary distribution to the value $q = n^{2.484+o(1)}$.

They also provide a concrete analysis, computing a bound on the modulus q for which the attacks exploiting the overstretched regime are more efficient than standard secret key recovery attacks. The authors of [12] run experiments which allow detecting the fatigue "point", which separates these two regimes.

The cryptanalysis carried out by [23] and [12] can be adapted for the iNTRU instances we consider, where the secret and the error both follow a Gaussian distribution. [12] also consider the case where the NTRU secret distributions are Gaussian. Indeed, in both cases, the cryptanalysis in the overstretched regime requires performing lattice reductions on sublattices of a NTRU-like lattice, in order to retrieve the very dense sublattice. We leave the detailed adaptation of this attack to iNTRU and its experimental study for future work. For our concrete parameters analysis, we took care to fall outside the range of parameters affected in the overstretched regime.

Timings. Our timings have been obtained on an Intel i7-8650U CPU running at 1.9 GHz, and then scaled at 4.2 GHz to compare ourselves with other schemes. Results are provided in Table 4. The use of the efficient gadget-based approximate trapdoor framework together with the iNTRU hardness assumption allows us to obtain efficient algorithms. The slowest of the 4 algorithms of the IBE2 scheme are the Setup and Extract algorithms, which correspond respectively to the approximate trapdoor generation and preimage sampling phases of the scheme. However, the Setup algorithm is usually not performed often, and the subroutine algorithms used by Extract for sampling are really modular, leaving the way for possible future improvements. Moreover, our Setup and Extract algorithms are competitive with other NTRU-based IBE (see Table 5 and Table 6).

Comparison with Related Works. We compare the IBE2 timings with the ones of [11] (re-implemented in [27]) and with [33], two IBE schemes whose security is based on the NTRU hardness assumption. Our comparison experiments were carried out using equivalent parameters sets between the different schemes. In particular, we have been careful to use equivalent module sizes and equivalent noise rates when performing Gaussian Sampling.

Table 4. Timings of the different operations for different values of n, given in ms from Setup to Decrypt.

(n, q)	$\lceil log_2(q) \rceil$	Setup	Extract	Encrypt	Decrypt	Bit security
(256, 1073741441)	30	1.01	2.13	0.39	0.03	64
(512, 1073741441)	30	2.12	3.79	0.74	0.06	115
(1024, 1073741441)	30	4.08	7.65	1.49	0.11	223
(256, 16777601)	25	0.78	1.66	0.23	0.02	35
(512, 16777601)	25	1.48	3.00	0.49	0.04	82
(1024, 16777601)	25	3.32	5.92	1.10	0.07	159

Table 5. Timings comparison of the different operations of the IBE scheme between this paper and [11] (the timings are extracted from the [27] article, scaled up to account for CPU differences) for different parameter sets. The timings are given in ms, except for the Setup algorithm from [11], which is given in seconds.

Scheme	$(n, \lceil log_2(q) \rceil)$	Security level	Setup	Extract	Encrypt	Decrypt
[11]	$(512, 26)$	< 80	3.84s	1.77	0.10	0.05
[11]	$(1024, 26)$	< 192	23.93s	6.95	0.27	0.09
This paper	$(512, 25)$	82	1.48	3.00	0.49	0.04
This paper	$(1024, 25)$	159	3.32	5.92	1.10	0.07

Table 6. Timings comparison of the different operations of the IBE scheme in this paper and in [33] for different sets of parameters in ms.

Scheme	$(n, \lceil log_2(q) \rceil)$	Security level	Setup	Extract	Encrypt	Decrypt
[33]	$(1024, 24)$	128	102	0.82	0.05	0.06
[33]	$(2048, 25)$	256	292	2.62	0.10	0.13
[33]	$(1024, 36)$	80	165	26.4	0.08	0.09
[33]	$(2048, 38)$	160	643	57.8	0.16	0.18
This paper	$(1024, 25)$	159	3.32	5.92	1.10	0.07
This paper	$(2048, 25)$	293	10.21	12.79	2.96	0.16
This paper	$(1024, 30)$	191	4.08	7.65	1.49	0.11
This paper	$(2048, 30)$	412	12.51	16.03	3.89	0.23

We observe that we obtain better timings for the Setup algorithm than [11] and [33] and for the Extract for some sets of parameters. Furthermore, our Decrypt algorithm is slightly faster than [11]. However, the Encrypt algorithm is less efficient than theirs. The use of binomial distribution improves the timings for encryption. An improvement can be obtained in our case by using the binomial distribution, but we need more samples in our case which still affects performance for encryption; we make $n(2 + m)$ calls to the integer Gaussian sampler while encryption in [11] and [33] can make use of only $3n$ binomial sampling calls. As in [33], our Extract algorithm is slower than the Sign algorithm from the Falcon signature scheme [13]. Note also that for a similar security level, Falcon can use smaller parameters than us.

We obtain an overhead in terms of parameters sizes compared to [11] and [33], but the trapdoor generations rely on a different paradigm. Results on public and secret sizes are provided in Appendix D. Nonetheless, as stated in [9], the use of approximate trapdoors instead of exact ones helps us to reduce the sizes of the public key and signatures by up to two times. Therefore, as expected, our obtained sizes for the master and the users' private keys and the ciphertexts are close to the ones of [16] whose signature scheme relies on the same paradigm as the IBE2 scheme.

Acknowledgements. This work is supported by the ANR ASTRID project AMIRAL (ANR-21-ASTR-0016).

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_28
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**, 169–203 (2015)
3. Alkım, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: USENIX Security Symposium (2016)
4. Bert, P., Eberhart, G., Prabel, L., Roux-Langlois, A., Sabt, M.: Implementation of lattice trapdoors on modules and applications. In: Cheon, J.H., Tillich, J.-P. (eds.) PQCrypto 2021 2021. LNCS, vol. 12841, pp. 195–214. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81293-5_11
5. Bert, P., Fouque, P., Roux-Langlois, A., Sabt, M.: Practical Implementation of Ring-SIS/LWE Based Signature and IBE (2018)
6. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_13
7. Boudgoust, K., Jeudy, C., Roux-Langlois, A., Wen, W.: On the hardness of module learning with errors with short distributions. J. Cryptol. **28**(1), 1 (2023)
8. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 523–552. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_27
9. Chen, Y., Genise, N., Mukherjee, P.: Approximate trapdoors for lattices and smaller hash-and-sign signatures. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 3–32. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_1
10. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 360–363. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45325-3_32

11. Ducas, L., Lyubashevsky, V., Prest, T.: Efficient identity-based encryption over NTRU lattices. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 22–41. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_2
12. Ducas, L., van Woerden, W.: NTRU fatigue: how stretched is overstretched? In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13093, pp. 3–32. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92068-5_1
13. Fouque, P.-A., et al.: Fast- Fourier Lattice-based Compact Signatures over NTRU (2017)
14. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_3
15. Genise, N., Gentry, C., Halevi, S., Li, B., Micciancio, D.: Homomorphic encryption for finite automata. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11922, pp. 473–502. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_17
16. Genise, N., Li, B.: Gadget-based iNTRU lattice trapdoors. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) INDOCRYPT 2020. LNCS, vol. 12578, pp. 601–623. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65277-7_27
17. Genise, N., Micciancio, D.: Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 174–203. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_7
18. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC (2008)
19. Karney, C.F.F.: Sampling exactly from the normal distribution. ACM Trans. Math. Softw. **42**(1), 3:1–3:14 (2016)
20. Kirchner, P., Fouque, P.-A.: Revisiting lattice attacks on overstretched NTRU parameters. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 3–26. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_1
21. Lai, R.W.F., Cheung, H.K.F., Chow, S.S.M.: Trapdoors for ideal lattices with applications. In: Lin, D., Yung, M., Zhou, J. (eds.) Inscrypt 2014. LNCS, vol. 8957, pp. 239–256. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16745-9_14
22. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. DCC **75**(3), 565–599 (2015)
23. Lee, C., Wallet, A.: Lattice analysis on MiNTRU problem. Cryptology ePrint Archive, Paper 2020/230 (2020)
24. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43
25. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_3
26. Lyubashevsky, V., Seiler, G.: Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 204–224. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_8

27. McCarthy, S., Smyth, N., O'Sullivan, E.: A practical implementation of identity-based encryption over NTRU lattices. In: O'Neill, M. (ed.) IMACC 2017. LNCS, vol. 10655, pp. 227–246. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71045-7_12
28. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
29. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on gaussian measures. SIAM J. Comput. **37**, 267–302 (2007)
30. Pataki, G., Tural, M.: On sublattice determinants in reduced bases. In: arXiv preprint arXiv:0804.4014 (2008)
31. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_5
32. Zhang, S., Yu, Y.: Towards a Simpler Lattice Gadget Toolkit. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography – PKC 2022. PKC 2022. LNCS, vol. 13177, pp. 498–520. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97121-2_18
33. Zhao, R.K., McCarthy, S., Steinfeld, R., Sakzad, A., O'Neill, M.: Quantumsafe HIBE: does it cost a Latte? ePrint Archive (2021)

Homomorphic Signatures for Subset and Superset Mixed Predicates and Its Applications

Masahito Ishizaka(✉) and Kazuhide Fukushima

KDDI Research, Inc., Saitama, Japan
{xma-ishizaka,ka-fukushima}@kddi.com

Abstract. In homomorphic signatures for subset predicates (HSSB), each message (to be signed) is a set. Any signature on a set M allows us to derive a signature on any subset $M' \subseteq M$. Its superset version, which should be called homomorphic signatures for superset predicates (HSSP), allows us to derive a signature on any superset $M' \supseteq M$. In this paper, we propose homomorphic signatures for subset and superset mixed predicates (HSSM) as a simple combination of HSSB and HSSP. In HSSM, any signature on a message of a set-pair (M, W) allows us to derive a signature on any (M', W') such that $M' \subseteq M$ and $W' \supseteq W$. We propose an original HSSM scheme which is unforgeable under the decisional linear assumption and completely context-hiding. We show that HSSM has various applications, which include disclosure-controllable HSSB, disclosure-controllable redactable signatures, (key-delegatable) superset/subset predicate signatures, and wildcarded identity-based signatures.

Keywords: Homomorphic signatures for subset and superset mixed predicates · Unforgeablity · Complete context-hiding · Decisional linear assumption

1 Introduction

$\mathcal{P}$-Homomorphic Signatures ($\mathcal{P}$-HS) [5]. In ordinary digital signatures, if a signed massage is partially altered, its signature immediately turns invalid. In $\mathcal{P}$-HS for a predicate $\mathcal{P} : \mathcal{M} \times \mathcal{M} \rightarrow 1/0$, any signature on any message M allows any user to derive a signature on any message M' satisfying the predicate, i.e., $1 \leftarrow \mathcal{P}(M, M')$. Strong context-hiding (SCH) [5] is a strong privacy (or unlinkability) related security notion, which guarantees that any signature derived from any signature which has been honestly generated distributes as a fresh signature directly generated by the secret-key. Complete context-hiding (CCH) [6] is a stronger notion, which guarantees that any signature derived from any *valid* signature (which might have been *dishonestly* generated) distributes as a signature generated by the secret-key. Redactable signatures (RS) [22] is a subclass of $\mathcal{P}$-HS. We can partially redact, i.e., black-out, a signed message

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 291–319, 2023.
https://doi.org/10.1007/978-3-031-35486-1_14

while retaining validity of the signature. In append-only signatures (AOS) [17], we can repeatedly add any message to the tail of a signed message. In quotable signatures [5,7], we can extract any substring from a signed message.

HS for Subset Predicates (HSSB) [5,6]. HSSB is also a subclass of $\mathcal{P}$-HS. Each message is a set. Any signature on a set M derives a signature on any subset $M' \subseteq M$. Ahn et al. [5] proposed a generic SCH-secure transformation into HSSB from attribute-based encryption (ABE) [8] satisfying a property that any secret-key for an attribute-set A allows us to derive a perfectly re-randomized secret-key for any subset $A' \subseteq A$. Attrapadung et al. [6] proposed an HSSB scheme which is CCH-secure and unforgeable under the decisional linear (DLIN) and q-simultaneous flexible paring (SFP) assumptions w.r.t. a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, built by the Groth-Sahai non-interactive witness indistinguishable (GS NIWI) proof [15], Waters signatures [23] and Abe-Haralambiev-Ohkubo structure-preserving signatures (AHO SPS) [3,4]. In key-generation, a signer generates a long-term AHO key-pair. When signing a message M, the signer generates a one-time Waters key-pair (g^x, x), where x is chosen uniformly at random from $\mathbb{Z}_p$, i.e., $x \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$, and g is a generator of $\mathbb{G}$, then generates an AHO signature on the Waters public-key g^x. The signer generates a Waters signature on every $m \in M$, then generates GS proofs that the AHO signature is valid and every Waters signature on $m \in M$ is valid.

HS for Superset Predicates (HSSP) [20]. HSSP is the superset counterpart of HSSB. It is called history-hiding append-only signatures (H2AOS) in [20]. Any signature on a set M derives a signature on any superset $M' \supseteq M$. Libert et al. [20] proposed a CCH-secure HSSP scheme based on the similar technique to the Attrapadung et al.'s HSSB scheme. It is built by an arbitrary SPS scheme satisfying unforgeability against extended random messages attacks [2], GS NIWI proof [15], and Boneh-Lynn-Shacham (BLS) signatures [11] instantiated by the Waters programmable hash function $H_{\mathbb{G}}$ [23]. In key-generation, a signer generates a long-term key-pair of the SPS scheme. When the signer signs a message W, she generates a fresh one-time BLS key-pair (Y, y), where $Y := g^y$ with $y \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$. The signer divides the BLS secret-key y into $|W|$ number of shares by using $|W|$-out-of-$|W|$ secret sharing. Specifically, she randomly chooses $\{\gamma_w \in \mathbb{Z}_p\}_{w \in W}$ s.t. $\sum_{w \in W} \gamma_w = y \pmod{p}$. For each $w \in W$, she computes $(\sigma_{w,1}, \sigma_{w,2}) := (H_{\mathbb{G}}(w)^{\gamma_w}, g^{\gamma_w}) \in \mathbb{G}^2$, where $\sigma_{w,1}$ is a BLS signature on w under the *pseudo* BLS public-key $\sigma_{w,2}$. The signer also generates a SPS signature $(\theta_1, \cdots, \theta_{l_{sps}}) \in \mathbb{G}^{l_{sps}}$ on $Y \in \mathbb{G}$. The signer finally generates a GS proof that (1) *the SPS signature on* Y *is valid*, (2) *the BLS signature* $\sigma_{w,1}$ *on* w *under the public-key* $\sigma_{w,2}$ *is valid for each* $w \in W$ and (3) $\prod_{w \in W} \sigma_{w,2} = Y$.

1.1 Our Contributions

HS for Subset and Superset Mixed Predicates (HSSM). We formally define HSSM as a simple combination of HSSB and HSSP. Any signature on a message of a set-pair (M, W) allows us to derive a signature on any (M', W') such that $M' \subseteq M$

and $W' \supseteq W$. As $\mathcal{P}$-HS [5,6], we define unforgeability and strong/complete context-hiding (SCH/CCH). Unforgeability is a security notion required for any $\mathcal{P}$-HS. We emphasize the importance of SCH and CCH by using the following example similar to an example in [12] used to emphasize the importance of unlinkability for sanitizable signatures. Given an honestly-generated signature σ on a set-pair (M, W), we honestly derive signatures σ_1 and σ_2 on (M_1, W_1) and (M_2, W_2), respectively, then open only the derived signatures to the public. If the HSSM scheme is SCH, both σ_1 and σ_2 are independent of σ and the link between σ_1 and σ_2 is unseen. Otherwise, it is possible that the link is seen. In this case, the following two types of private information are leaked, (1) *M contains at least $M_1 \cup M_2$ as a subset*, and (2) *every element in $W_1 \cap W_2$ is older data than any element in $W_1 \setminus (W_1 \cap W_2)$ and in $W_2 \setminus (W_1 \cap W_2)$*. If the HSSM scheme is CCH, even when the signatures σ, σ_1 and σ_2 have been dishonestly generated, the same security is guaranteed. Thus, CCH is more desirable. If the number of the derived signatures is $n > 2$, a non-SCH HSSB scheme can cause a more serious problem, e.g., we can guess that M contains at least $\bigcup_{i=1}^{n} M_i$.

Our HSSM Scheme. Our HSSM scheme is a simple combination of the Attrapadung et al.'s HSSB scheme [6] and the Libert et al.'s HSSP schemes [20]. As the underlying SPS scheme, we use the AHO SPS scheme [3,4] with message space $\mathbb{G}^2$. On signature generation, a fresh one-time Waters key-pair (X, x) and BLS key-pair (Y, y) are generated, where $X := g^x$ and $Y := g^y$ with $x, y \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$. An AHO signature $(\theta_1, \cdots, \theta_7)$ on $(X, Y) \in \mathbb{G}^2$ is generated. As the HSSB scheme [6], for each $m \in M$, a Waters signature $(\sigma_{m,1}, \sigma_{m,2})$ on m is generated. As the HSSP scheme [20], the original BLS secret-key $y \in \mathbb{Z}_p$ is divided into $|W|$ number of shares $\{\gamma_w \in \mathbb{Z}_p\}_{w \in W}$ by $|W|$-out-of-$|W|$ secret sharing, then for each $w \in W$, a BLS signature $\sigma_{w,1}$ on w under the pseudo BLS public-key $g^{\gamma_w} (=: \sigma_{w,2})$ is generated. Finally, the GS proof is properly generated.

We show that HSSM has following five applications.

Application 1: Disclosure-Controllablwe (DC) HSSB. In the ordinary HSSB [5,6], any sub-message $m \in M$ can be deleted anytime. For some realistic applications of HSSB, there might be a case where we would like to make some sub-messages undeletable. For instance, HSSB can be used to prove one's credentials. A person named Alice is given an HSSM signature on a message M including her identifiable information (e.g., name, her birth date) and all of her credentials. When she proves that she has all of the required credentials to an organization, she might want to hide all of the unrequired credentials. In this application, her identifiable information is usually not deleted in any situation. Even in the ordinary HSSB, when and only when the original signature is generated, some sub-messages can be designated as undeletable. For instance, for a message $M = \{A, B, C\}$, if the signer wants to make B undeletable, she produces a special sub-message D stating that B is undeletable and signs the updated set $M' := M \cup \{D\}$. This simple approach has a problem that the sub-message D itself can be deleted. To resolve it, we prepend bit 1 to D and bit 0 to the others, and we make an agreement in advance that every message must have one sub-message starting with bit 1. In our DCHSSB, any deletable sub-message can be changed to undeletable anytime. The change is one-way, which means we cannot make any undeletable sub-message deletable. If every sub-message goes undeletable at some point, the message is finalized. We show that DCHSSB can be transformed from HSSM. More specifically, DCHSSB is a subclass of HSSM. From our HSSM scheme, we obtain a DCHSSB scheme secure under the DLIN assumption. To the best of our knowledge, our DCHSSB scheme is the first one which is adaptively unforgeable and strongly context-hiding under the standard assumptions.

Application 2: Disclosure-Controllable RS. In redactable signatures (RS) with maximal number of sub-messages $N \in \mathbb{N}$, each message M is an ordered list in the form of $(m_1, \cdots, m_n)$ for some $n \in [1, N]$, where $m_i \in \{0,1\}^L \cup \{*\}$. Each (non-redacted) sub-message $m_i (\neq *)$ can be changed to $*$, which means it has been redacted, i.e., blacked out. In the ordinary RS, any (non-redacted) sub-message can be redacted anytime. In disclosure-controllable RS (DCRS), any sub-message which is *non-redacted* and *redactable* can be *unredactable*. Specifically, each sub-message has one of the following three states, namely **S1**: not redacted yet and redactable, **S2**: already redacted, and **S3**: not redacted yet and unredactable. Any state only transitions from **S1** to **S2** or from **S1** to **S3**. If every sub-message goes in **S2** or **S3**, the message is finalized. We show that DCRS can be transformed from DCHSSB. From our DCHSSB scheme, we obtain a DCRS scheme secure under the DLIN assumption. As our DCHSSB scheme explained earlier, our DCRS scheme is the first one which is adaptively unforgeable and strongly context-hiding under the standard assumptions.

Application 3: (Key-Delegatable) Subset Predicate Signatures (SBPS). Subset predicate signatures is the digital signature analogue of subset predicate encryption [16]. A secret-key associated with a set $X \in 2^{\{0,1\}^L}$ succeeds in generating a signature on a message associated with any superset $Y \supseteq X$. Our SBPS has key-delegatability, which means that a secret-key for X generates a new secret-key for any superset $X' \supseteq X$. As attribute-based signatures [9,21], we define unforgeability and signer-privacy. The latter security guarantees that any signature with a set Y has no more information about the signer's set X than the fact that $X \subseteq Y$. Identity-based ring signatures (IBRS) [24] is a subclass of SBPS because IBRS is identical to SBPS with the following two restrictions, (1) key-delegation is not allowed and (2) cardinality of the set X is fixed to 1. HSSP can be transformed into IBRS as shown in [20]. In fact, it can be transformed into the stronger primitive SBPS.

Application 4: (Key-Delegatable) Superset Predicate Signatures (SPPS). SPPS is the dual primitive of SBPS. A secret-key associated with a set $X \in 2^{\{0,1\}^L}$ generates a signature associated with any subset $Y \subseteq X$ and generates another secret-key associated with any subset $X' \subseteq X$. We show that SPPS can be transformed from HSSM in a simple and efficient manner. Actually, SPPS can be transformed from a weaker primitive HSSB. However, its transformation itself is somewhat complicated. Moreover, if we instantiate it by any one of the existing SCH-secure HSSB scheme, an inefficient SPPS scheme whose secret-key and signature lengths increase linearly with the message length is obtained.

Application 5: Wildcarded Identity-Based Signatures (WIBS). WIBS is the digital signatures analogue of wildcarded identity-based encryption [1]. Each secret-key is associated with an identity $X = (x_1, \cdots, x_n) \in (\{0,1\}^L)^n$ for some $L, n \in \mathbb{N}$, and it succeeds in generating a signature associated with any wildcarded identity $Y = (y_1, \cdots, y_n) \in (\{0,1\}^L \cup \{*\})^n$ s.t. $y_i \neq * \implies x_i = y_i$ for all $i \in [1, n]$. We show that WIBS can be transformed from HSSM efficiently.

Paper Organization. In Sect. 2, we explain notions used in this paper, and define symmetric bilinear paring and some computational assumptions. In Sect. 3, we define general $\mathcal{P}$-HS, then show that HSSM is a subclass of $\mathcal{P}$-HS. In Sect. 4, we propose our HSSM schemes. In Sect. 5, we present the applications of HSSM.

2 Preliminaries

Notations. For $\lambda \in \mathbb{N}$, 1^λ denotes a security parameter. A function $f : \mathbb{N} \to \mathbb{R}$ is negligible if for every $c \in \mathbb{N}$, there exists $x_0 \in \mathbb{N}$ such that for every $x \geq x_0$, $f(x) \leq x^{-c}$. We parse a binary string $x \in \{0,1\}^L$ as $x[L-1] || \cdots || x[0]$, where $x[i] \in \{0,1\}$ for all $i \in [0, L-1]$. PPT is the abbreviation of probabilistic polynomial-time. For a set A, $a \overset{\mathrm{U}}{\longleftarrow} A$ means that an element a is chosen uniformly at random from A. For a set A, $|A|$ denotes its cardinality.

$\mathcal{G}$ takes a security parameter 1^λ with $\lambda \in \mathbb{N}$ and outputs a group description $(p, \mathbb{G}, \mathbb{G}_T, e, g)$. p is a prime with length λ. $\mathbb{G}$ and $\mathbb{G}_T$ are multiplicative groups with order p. g is a generator of $\mathbb{G}$. $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently-computable function which satisfies both of the following conditions, namely (1) bilinearity: for any $a, b \in \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$, and (2) non-degeneracy: $e(g, g) \neq 1_{\mathbb{G}_T}$, where $1_{\mathbb{G}_T}$ denotes the unit element of $\mathbb{G}_T$.

We define three computational hardness assumptions.

Definition 1. *The computational Diffie-Hellman (CDH) assumption holds on the group $\mathbb{G}$ if for every PPT $\mathcal{A}$, its advantage $\mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{A},\mathbb{G}}(\lambda) := \Pr[g^{ab} \leftarrow \mathcal{A}(g, g^a, g^b)]$ (with $a, b \overset{\mathrm{U}}{\longleftarrow} \mathbb{Z}_p$) is negligible.*

Definition 2. *The decisional linear (DLIN) assumption holds on the group $\mathbb{G}$ if for every PPT $\mathcal{A}$, its advantage $\mathtt{Adv}^{\mathtt{DLIN}}_{\mathcal{A},\mathbb{G}}(\lambda) := |\Pr[1 \leftarrow \mathcal{A}(g^a, g^b, g^{ab}, g^{bd}, g^{c+d})]| - \Pr[1 \leftarrow \mathcal{A}(g^a, g^b, g^{ab}, g^{bd}, g^z)]$ (with $a, b, c, d, z \overset{\mathrm{U}}{\longleftarrow} \mathbb{Z}_p$) is negligible.*

Definition 3. *The q-simultaneous flexible pairing (q-SFP) problem [4] relative to the group $\mathbb{G}$ is given $g_z, h_z, g_r, h_r, a, \tilde{a}, b, \tilde{b} \in \mathbb{G}$ and q tuples $\{(z_j, r_j, s_j, t_j, u_j, v_j, w_j) \in \mathbb{G}^7\}_{j=1}^q$ such that for each $j \in [1, q]$,*

$$e(a, \tilde{a}) = e(g_z, z_j) \cdot e(g_r, r_j) \cdot e(s_j, t_j), \quad e(b, \tilde{b}) = e(h_z, z_j) \cdot e(h_r, u_j) \cdot e(v_j, w_j), \quad (1)$$

to find a new tuple $(z^\star, r^\star, s^\star, t^\star, u^\star, v^\star, w^\star) \in \mathbb{G}^7$ *satisfying the conditions* (1) *and the condition* $z^\star \notin \{1_\mathbb{G}, z_1, \cdots, z_q\}$*. The q-SFP assumption holds if for any PPT* $\mathcal{A}$*, its advantage* $\mathtt{Adv}^{q\text{-}SFP}_{\mathcal{A},\mathbb{G}}(\lambda)$ *to solve the above problem is negligible.*

3 HS for Subset and Superset Mixed Predicates (HSSM)

We firstly define Homomorphic signatures for a general predicate $\mathcal{P} : \mathcal{M} \times \mathcal{M} \times 1/0$ (abbreviated as $\mathcal{P}$-HS) [5], then define HS for subset and superset mixed predicates (HSSM) as a subclass of $\mathcal{P}$-HS.

Syntax. $\mathcal{P}$-HS consists of the following four polynomial-time algorithms. $\mathtt{Ver}$ is deterministic and the others are probabilistic.

Key-Generation $\mathtt{KGen}$**:** This algorithm takes 1^λ, then outputs a public-key *pk* and a secret-key *sk*. $(pk, sk) \leftarrow \mathtt{KGen}(1^\lambda)$

Signing $\mathtt{Sig}$**:** Take a secret-key *sk* and a message $M \in \mathcal{M}$, then output a signature σ. $\sigma \leftarrow \mathtt{Sig}(sk, M)$

Derivation $\mathtt{Derive}$**:** Take a public-key *pk*, a message M, a signature σ and a message $M' \in \mathcal{M}$, then output a signature σ'. $\sigma' \leftarrow \mathtt{Derive}(pk, M, \sigma, M')$

Verification $\mathtt{Ver}$**:** Take a public-key *pk*, a message $M \in \mathcal{M}$ and a signature σ, then output 1 (meaning *accept*) or 0 (*reject*). $1/0 \leftarrow \mathtt{Ver}(pk, M, \sigma)$

Every $\mathcal{P}$-HS scheme must be correct. A $\mathcal{P}$-HS scheme is correct if for any $\lambda \in \mathbb{N}$, any $(pk, sk) \leftarrow \mathtt{KGen}(1^\lambda)$, any $M \in \mathcal{M}$ and any $M' \in \mathcal{M}$ s.t. $1 \leftarrow \mathcal{P}(M, M')$, both of the following two conditions are satisfied, namely (1) $1 \leftarrow \mathtt{Ver}(pk, M, \mathtt{Sig}(sk, M))$ and (2) $1 \leftarrow \mathtt{Ver}(pk, M, \mathtt{Derive}(pk, M, \mathtt{Sig}(sk, M), M'))$.

Security. As security, we define unforgeability and context-hiding. As unforgeability notions, we define unforgeability (UNF) and weak unforgeability (wUNF) with the following experiments.

```
$\boldsymbol{Expt}^{\mathtt{UNF}}_{\Sigma_{\text{P-HS}},\mathcal{A}}(1^\lambda)$:   //$\boldsymbol{Expt}^{\mathtt{wUNF}}_{\Sigma_{\text{P-HS}},\mathcal{A}}$
  1. Generate $(pk, sk) \leftarrow \mathtt{KGen}(1^\lambda)$. Initialize two tables $Q, Q' := \emptyset$
  2. $(\sigma^*, M^* \in \mathcal{M}) \leftarrow \mathcal{A}^{\mathfrak{Sign},\mathfrak{Derive},\mathfrak{Reveal}}(pk)$
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  - $\mathfrak{Sign}(M \in \mathcal{M})$:
      $\sigma \leftarrow \mathtt{Sig}(sk, M)$. Choose an unused handle $h \in \mathcal{H}$.
      $Q := Q \cup \{(h, M, \sigma)\}$. Rtrn $h$
  - $\mathfrak{Derive}(h \in \mathcal{H}, M' \in \mathcal{M})$:
      Rtrn $\bot$. Rtrn $\bot$ if $\nexists(M, \sigma)$ s.t. $(h, M, \sigma) \in Q \land 1 \leftarrow \mathcal{P}(M, M')$.
      $\sigma' \leftarrow \mathtt{Derive}(pk, M, \sigma, M')$. Choose an unused handle $h' \in \mathcal{H}$.
      $Q := Q \cup \{(h', M', \sigma')\}$. Rtrn $h'$
  - $\mathfrak{Reveal}(h \in \mathcal{H})$:
      Rtrn $\bot$ if $\nexists(M, \sigma)$ s.t. $(h, M, \sigma) \in Q$. Update $Q' := Q' \cup \{M\}$. Rtrn $\sigma$
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  3. Rtrn 1 if (1) $1 \leftarrow \mathtt{Ver}(pk, M^*, \sigma^*)$ and (2) $0 \leftarrow \mathcal{P}(M, M^*)$ for all $M \in Q'$.
  4. Rtrn 0
```

In the experiment for `UNF`, the PPT adversary $\mathcal{A}$ receives an honestly generated public-key pk, adaptively accesses three oracles, namely signing, derivation and (signature-)revelation, then outputs a forged signature σ^* on M^*. The grey command **Rtrn** $\perp$. in the derivation oracle is considered only in the experiment for `wUNF`, which means that the derivation oracle is useless because it always returns $\perp$. `UNF` is defined as follows and `wUNF` is analogously defined.

Definition 4. *A scheme $\Sigma_{\text{P-HS}}$ is `UNF` if for any $\lambda \in \mathbb{N}$ and any PPT algorithm $\mathcal{A}$, its advantage $\boldsymbol{Adv}^{\mathit{UNF}}_{\Sigma_{\text{P-HS}},\mathcal{A}}(\lambda) := \Pr[1 \leftarrow \boldsymbol{Expt}^{\mathit{UNF}}_{\Sigma_{\text{P-HS}},\mathcal{A}}(1^\lambda)]$ is negligible.*

As privacy (or unlinkability) related security notions, we define strong context-hiding (`SCH`) [5] and complete context-hiding (`CCH`) [6].

Definition 5. *A $\mathcal{P}$-HS scheme is `CCH` (resp. `SCH`) if for any $\lambda \in \mathbb{N}$, any $(pk, sk) \leftarrow \mathtt{KGen}(1^\lambda)$, any message $M \in \mathcal{M}$, any valid signature σ s.t. $1 \leftarrow \mathtt{Ver}(pk, M, \sigma)$ (resp. any honestly-generated signature $\sigma \leftarrow \mathtt{Sig}(sk, M)$), any message $M' \in \mathcal{M}$ s.t. $1 \leftarrow \mathcal{P}(M, M')$, the following two distributions are identical, namely (1) $\{sk, \sigma, \mathtt{Derive}(pk, M, \sigma, M')\}$ and (2) $\{sk, \sigma, \mathtt{Sig}(sk, M')\}$.*

It is obvious that, for any $\mathcal{P}$-HS scheme, `CCH` implies `SCH`, and the conjunction of `SCH` and `wUNF` implies `UNF`.

HSSM as a Subclass of $\mathcal{P}$-HS. In HSSM, each message is a pair of sets $(M, W) \in 2^{\{0,1\}^L} \times 2^{\{0,1\}^L}$ for an integer $L \in \mathbb{N}$. Predicate $\mathcal{P}_{\mathtt{mixed}}$ for HSSM takes two messages $(M, W), (M', W') \in 2^{\{0,1\}^L} \times 2^{\{0,1\}^L}$, then outputs 1 if (1) *M' is a subset of M, i.e., $M' \subseteq M$*, and (2) *W' is a superset of W, i.e., $W' \supseteq W$*. HSSM is a simple combination of HS for subset predicates (HSSB) [5,6] and HS for superset predicates (HSSP) originally called history-hiding append-only signatures in [20].

4 Our HSSM Schemes

Groth-Sahai Non-Interactive Witness-Indistinguishable (GS NIWI) Proof [15]. In the GS NIWI proof system, based on a symmetric bilinear pairing $e : \mathbb{G}^2 \rightarrow \mathbb{G}_T$ with groups $\mathbb{G}, \mathbb{G}_T$ whose order is a prime p, a CRS consists of three vectors $\vec{f}_1, \vec{f}_2, \vec{f}_3 \in \mathbb{G}^3$, where $\vec{f}_1 = (f_1, 1, g)$ and $\vec{f}_2 = (1, f_2, g)$ with $f_1, f_2 \in \mathbb{G}$. A commitment $\vec{C}$ to an element $\mathcal{X} \in \mathbb{G}$ is given as $(1, 1, \mathcal{X}) \cdot \vec{f}_1^{\,r} \cdot \vec{f}_2^{\,s} \cdot \vec{f}_3^{\,t}$ with $r, s, t \overset{\text{U}}{\leftarrow} \mathbb{Z}_p$. The CRS is in one of the two settings. In the perfect soundness setting, where the vector $\vec{f}_3$ is chosen outside the span of the other vectors $\vec{f}_1$ and $\vec{f}_2$, any commitment is perfectly hiding. In the perfect witness-indistinguishability(WI) setting, the CRS satisfies $\vec{f}_3 = \vec{f}_1^{\,\xi_1} \cdot \vec{f}_2^{\,\xi_2}$ with $\xi_1, \xi_2 \in \mathbb{Z}_p$, and from any commitment $\vec{C} = (f_1^{r+\xi_1 t}, f_2^{s+\xi_2 t}, \mathcal{X} \cdot g^{r+s+t(\xi_1+\xi_2)})$ distributing identically to a ciphertext of the Boneh-Boyen-Shacham (BBS) encryption [10], the committed variable $\mathcal{X}$ is extracted by using $\beta_1 = \log_g(f_1)$ and $\beta_2 = \log_g(f_2)$. The two settings of CRS are hard to distinguish under the decisional linear (DLIN) assumption.

Given n variables $\mathcal{X}_1, \cdots, \mathcal{X}_n \in \mathbb{G}$, a prover computes a commitment $\vec{C}_{\mathcal{X}_i} \in \mathbb{G}^3$ to each variable $\mathcal{X}_i$ with randomness $r_i, s_i, t_i \overset{\mathrm{U}}{\longleftarrow} \mathbb{Z}_p$, then computes a proof for a pairing-product equation (PPE) in a general form of $\prod_{i=1}^{n} e(\mathcal{A}_i, \mathcal{X}_i) \cdot \prod_{i=1}^{n} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{X}_j)^{a_{ij}} = t_T$ with constants $\mathcal{A}_i \in \mathbb{G}$, $a_{ij} \in \mathbb{Z}_p$ and $t_T \in \mathbb{G}_T$. In this paper, we use only a simpler form of $\prod_{i=1}^{n} e(\mathcal{A}_i, \mathcal{X}_i) = t_T$. In this case, the proof $\vec{\pi} \in \mathbb{G}^3$ is simply $(\prod_{i=1}^{n} \mathcal{A}_i^{r_i}, \prod_{i=1}^{n} \mathcal{A}_i^{s_i}, \prod_{i=1}^{n} \mathcal{A}_i^{t_i})$. Each commitment $\vec{C}_{\mathcal{X}_i}$ is publicly and perfectly re-randomized by choosing $r'_i, s'_i, t'_i \overset{\mathrm{U}}{\longleftarrow} \mathbb{Z}_p$ then computing $\vec{C}'_{\mathcal{X}_i} := \vec{C}_{\mathcal{X}_i} \cdot \vec{f}_1^{r'_i} \cdot \vec{f}_2^{s'_i} \cdot \vec{f}_3^{t'_i}$. The proof $\vec{\pi}$ must be naturally updated to $\vec{\pi} \cdot (\prod_{i=1}^{n} \mathcal{A}_i^{r'_i}, \prod_{i=1}^{n} \mathcal{A}_i^{s'_i}, \prod_{i=1}^{n} \mathcal{A}_i^{t'_i})$.

Attrapadung et al.'s HSSB Scheme [6]. Their HSSB scheme is based on the GS proof [15], Waters signatures [23] (described in Appendix 3) and Abe-Haralambiev-Ohkubo (AHO) structure-preserving signatures (SPS) [3,4] (described in Appendix 4). In key-generation, a signer generates a long-term AHO key-pair. When signing a message $M \in 2^{\{0,1\}^L}$, the signer freshly generates a one-time Waters key-pair (X, x), where $X := g^x$ with $x \overset{\mathrm{U}}{\longleftarrow} \mathbb{Z}_p$. For each sub-message $m \in M$, the signer generates a Waters signature $(\sigma_{m,1}, \sigma_{m,2}) := (h^x \cdot H_{\mathbb{G}}(m)^{\chi_m}, g^{\chi_m}) \in \mathbb{G}^2$ under a public group element $h \in \mathbb{G}$ and a randomness $\chi_m \overset{\mathrm{U}}{\longleftarrow} \mathbb{Z}_p$. The signer also generates an AHO signature $(\theta_1, \cdots, \theta_7) \in \mathbb{G}^7$ on the Waters public-key $X \in \mathbb{G}$. Finally, the signer computes GS commitments to $X, \{\sigma_{m,1}\}_{m \in M}, \theta_1, \theta_2, \theta_5 \in \mathbb{G}$, then computes GS proofs that (1) *the AHO signature* $(\theta_1, \cdots, \theta_7)$ *on* X *is valid* and (2) *the Waters signature* $(\sigma_{m,1}, \sigma_{m,2})$ *on* m *under the public-key* X *is valid for each* $m \in M$. When a public party derives a signature on a subset $M' \subseteq M$, for each deleted sub-message $m \in M \setminus M'$, the commitment to $\sigma_{m,1}$, (non-committed) $\sigma_{m,2}$ and the GS proof related to this Waters signature are deleted from the original HSSB signature, then all of the remaining variables are perfectly re-randomized.

Libert et al.'s HSSP Scheme [20]. Libert et al.'s HSSP scheme is similar to the Attrapadung et al.'s HSSB scheme. It is based on an arbitrary SPS scheme satisfying unforgeability against extended random messages attacks [2], GS NIWI proof [15], and Boneh-Lynn-Shacham (BLS) signatures [11] instantiated by the Waters programmable hash function [23], i.e., $H_{\mathbb{G}}$. In key-generation, a signer generates a long-term key-pair of the SPS scheme. When signing a message $W \in 2^{\{0,1\}^L}$, the signer generates a one-time BLS key-pair (Y, y), where $Y := g^y$ with $y \overset{\mathrm{U}}{\longleftarrow} \mathbb{Z}_p$. The signer divides the BLS secret-key y into $|W|$ number of shares by using $|W|$-out-of-$|W|$ secret sharing[1]. Specifically, she uniform-randomly chooses $\{\gamma_w \in \mathbb{Z}_p\}_{w \in W}$ satisfying that $\sum_{w \in W} \gamma_w = y \pmod{p}$. For each $w \in W$, she computes $(\sigma_{w,1}, \sigma_{w,2}) := (H_{\mathbb{G}}(w)^{\gamma_w}, g^{\gamma_w}) \in \mathbb{G}^2$, where the first element $\sigma_{w,1}$ is a BLS signature on w under the *pseudo* BLS public-key $\sigma_{w,2}$. The signer also generates a SPS signature $(\theta_1, \cdots, \theta_{l_{sps}}) \in \mathbb{G}^{l_{sps}}$ on $Y \in \mathbb{G}$. Finally, the signer computes GS commitments to $Y, \{\sigma_{w,1}, \sigma_{w,2}\}_{w \in W}, \theta_1, \cdots, \theta_{l_{sps}} \in \mathbb{G}$, then computes GS proofs that (1) *the SPS signature* $(\theta_1, \cdots, \theta_{l_{sps}})$ *on* Y *is*

[1] $|W|$ denotes cardinality of the set W.

valid, (2) *the BLS signature* $\sigma_{w,1}$ *on* w *under the pseudo public-key* $\sigma_{w,2}$ *is valid for each* $w \in W$ and (3) $\prod_{w \in W} \sigma_{w,2} = Y$. When a public user derives a signature on a superset $W' \supseteq W$, $\{\gamma_w \in \mathbb{Z}_p\}_{w \in W'}$ s.t. $\sum_{w \in W'} \gamma_w = 0 \pmod{p}$ are randomly chosen. For each *added* element $w \in W' \setminus W$, we compute $(\sigma_{w,1}, \sigma_{w,2}) := (H_{\mathbb{G}}(w)^{\gamma'_w}, g^{\gamma'_w})$ then compute GS commitments to them. For each *inherited* element $w \in W' \cap W$, we update the committed variables $\sigma_{w,1}, \sigma_{w,2} \in \mathbb{G}$ to $\sigma_{w,1} \cdot H_{\mathbb{G}}(w)^{\gamma'_w}$ and $\sigma_{w,2} \cdot g^{\gamma'_w}$, respectively. The GS proof for the relation $\prod_{w \in W} \sigma_{w,2} = Y$ is properly updated and all of the GS commitments and proofs are perfectly re-randomized.

4.1 Construction

Our HSSM scheme is a simple combination of the Attrapadung et al.'s HSSB and Libert et al.'s HSSP schemes. As the underlying SPS scheme, we also use the AHO SPS scheme [3,4] with message space $\mathbb{G}^2$. On signature generation, a fresh one-time Waters and BLS key-pairs (X, x), (Y, y) are generated, where $X := g^x$ and $Y := g^y$ with $x, y \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$. An AHO signature $(\theta_1, \cdots, \theta_7)$ on $(X, Y) \in \mathbb{G}^2$ is generated. For each $m \in M$, a Waters signature $(\sigma_{m,1}, \sigma_{m,2})$ on m is generated. The original BLS secret-key $y \in \mathbb{Z}_p$ is divided into $|W|$ number of shares $\{\gamma_w \in \mathbb{Z}_p\}_{w \in W}$ by $|W|$-out-of-$|W|$ secret sharing, then for each $w \in W$, a BLS signature $\sigma_{w,1}$ on w under the pseudo BLS public-key $g^{\gamma_w} (=: \sigma_{w,2})$ is generated. Finally, the GS commitments and proofs are properly generated.

For any element $Z \in \mathbb{G}$, $\iota(Z)$ denotes $(1, 1, Z) \in \mathbb{G}^3$. For any $h, g_1, g_2, g_3 \in \mathbb{G}$, $E(h, (g_1, g_2, g_3))$ denotes $(e(h, g_1), e(h, g_2), e(h, g_3)) \in \mathbb{G}_T^3$. Our HSSM scheme is formally described as follows.

$\texttt{KGen}(1^\lambda, L)$**:** Choose bilinear groups $(\mathbb{G}, \mathbb{G}_T)$ whose order is a prime p of bit length λ. g denotes a generator of $\mathbb{G}$. Conduct the following three steps.

1. Generate the public parameter of the Waters signatures [23] by choosing $h, u', u_0, \cdots, u_{L-1} \overset{\text{U}}{\longleftarrow} \mathbb{G}$. The Waters programmable hash function $H_{\mathbb{G}} : \{0,1\}^L \to \mathbb{G}$ takes a sub-message $m \in \{0,1\}^L$, being parsed as $m[L-1] \parallel \cdots \parallel m[0]$, then outputs $u' \prod_{i=0}^{L-1} u_i^{m[i]} \in \mathbb{G}$.
2. Generate a key-pair $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$ of the AHO SPS [3,4]. Parse it as $((g_r, h_r, g_z, h_z, g_1, h_1, g_2, h_2, A, B), (\alpha_a, \alpha_b, \gamma_z, \delta_z, \gamma_1, \delta_1, \gamma_2, \delta_2))$.
3. Generate a CRS $\boldsymbol{f} = (\vec{f}_1, \vec{f}_2, \vec{f}_3)$ for the perfect WI setting of the GS NIWI proof [15]. Concretely, $\vec{f}_1 := (f_1, 1, g)$, $\vec{f}_2 := (1, f_2, g)$ and $\vec{f}_3 := \vec{f}_1^{\xi_1} \cdot \vec{f}_2^{\xi_2} \cdot (1, 1, g)^{-1}$, where $f_1, f_2 \overset{\text{U}}{\longleftarrow} \mathbb{G}$ and $\xi_1, \xi_2 \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$.

Output (pk, sk), where $pk := (\mathbb{G}, \mathbb{G}_T, g, h, u', \{u_i\}_{i=0}^{L-1}, pk_{\mathsf{s}}, \boldsymbol{f})$ and $sk := sk_{\mathsf{s}}$.

$\texttt{Sig}(sk, M \in 2^{\{0,1\}^L}, W \in 2^{\{0,1\}^L})$**:** Firstly, conduct the following six steps.

1. Choose $x, y \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$. Let $X := g^x$ and $Y := g^y$.
2. Generate an AHO signature $\sigma_{\mathsf{s}} = (\theta_1, \cdots, \theta_7)$ on a message $(X, Y) \in \mathbb{G}^2$.
3. For each $m \in M$, generate a Waters signature on m, i.e., $(\sigma_{m,1}, \sigma_{m,2}) := (h^x \cdot H_{\mathbb{G}}(m)^{\chi_m}, g^{\chi_m})$ with $\chi_m \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$.

4. Choose $\{\gamma_w \in \mathbb{Z}_p\}_{w \in W}$ satisfying $\sum_{w \in W} \gamma_w = y \pmod{p}$ uniformly at random. Then for each $w \in W$, compute $(\sigma_{w,1}, \sigma_{w,1}) := (H_{\mathbb{G}}(w)^{\gamma_w}, g^{\gamma_w})$.
5. Compute GS commitments for all of the group elements X, Y, θ_1, θ_2, θ_5, $\{\sigma_{m,1}\}_{m \in M}$ and $\{\sigma_{w,1}, \sigma_{w,2}\}_{w \in W}$. The commitments are denoted by $\vec{C}_X, \vec{C}_Y, \vec{C}_{\theta_1}, \vec{C}_{\theta_2}, \vec{C}_{\theta_5}, \vec{C}_{\sigma_{m,1}}, \vec{C}_{\sigma_{w,1}}, \vec{C}_{\sigma_{w,2}}$, respectively. Specifically, for each committed element $Z \in \mathbb{G}$, the commitment $\vec{C}_Z$ is computed as $\iota_{\mathbb{G}}(Z) \cdot \vec{f}_1^{\,r_Z} \cdot \vec{f}_2^{\,s_Z} \cdot \vec{f}_3^{\,t_Z}$ with $r_Z, s_Z, t_Z \xleftarrow{\text{U}} \mathbb{Z}_p$.
6. Compute GS proofs for all of the following PPEs.

$$A \cdot e(\theta_3, \theta_4)^{-1} = e(g_z, \theta_1) \cdot e(g_r, \theta_2) \cdot e(g_1, X) \cdot e(g_2, Y) \tag{2}$$

$$B \cdot e(\theta_6, \theta_7)^{-1} = e(h_z, \theta_1) \cdot e(h_r, \theta_5) \cdot e(h_1, X) \cdot e(h_2, Y) \tag{3}$$

$$e(\sigma_{m,1}, g) = e(X, h) \cdot e(\sigma_{m,2}, H_{\mathbb{G}}(m)) \quad (\textbf{For each } m \in M) \tag{4}$$

$$e(\sigma_{w,1}, g) = e(\sigma_{w,2}, H_{\mathbb{G}}(w)) \quad (\textbf{For each } w \in W) \tag{5}$$

$$e(Y, g) = \prod_{w \in W} e(\sigma_{w,2}, g) \tag{6}$$

The proofs are denoted by $\vec{\pi}_A, \vec{\pi}_B, \vec{\pi}_m, \vec{\pi}_w$ and $\vec{\pi}_{sum}$, respectively, and computed as follows.

$$\begin{aligned}
\vec{\pi}_A &:= (g_z^{r_{\theta_1}} g_r^{r_{\theta_2}} g_1^{r_X} g_2^{r_Y}, g_z^{s_{\theta_1}} g_r^{s_{\theta_2}} g_1^{s_X} g_2^{s_Y}, g_z^{t_{\theta_1}} g_r^{t_{\theta_2}} g_1^{t_X} g_2^{t_Y}) \\
\vec{\pi}_B &:= (h_z^{r_{\theta_1}} h_r^{r_{\theta_2}} h_1^{r_X} h_2^{r_Y}, h_z^{s_{\theta_1}} h_r^{s_{\theta_2}} h_1^{s_X} h_2^{s_Y}, h_z^{t_{\theta_1}} h_r^{t_{\theta_2}} h_1^{t_X} h_2^{t_Y}) \\
\vec{\pi}_m &:= (g^{r_{\sigma_{m,1}}} \cdot h^{-r_X}, g^{s_{\sigma_{m,1}}} \cdot h^{-s_X}, g^{t_{\sigma_{m,1}}} \cdot h^{-t_X}) \\
\vec{\pi}_w &:= (g^{r_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-r_{\sigma_{w,2}}}, g^{s_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-s_{\sigma_{w,2}}}, g^{t_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-t_{\sigma_{w,2}}}) \\
\vec{\pi}_{sum} &:= (g^{r_Y - \sum_{w \in W} r_{\sigma_{w,2}}}, g^{s_Y - \sum_{w \in W} s_{\sigma_{w,2}}}, g^{t_Y - \sum_{w \in W} t_{\sigma_{w,2}}})
\end{aligned}$$

Finally, output a signature σ which is

$$\begin{pmatrix} \vec{C}_X, \vec{C}_Y, \{\vec{C}_{\theta_i}\}_{i \in \{1,2,5\}}, \{\theta_i\}_{i \in \{3,4,6,7\}}, \vec{\pi}_A, \vec{\pi}_B, \\ \{\vec{C}_{\sigma_{m,1}}, \sigma_{m,2}, \vec{\pi}_m\}_{m \in M}, \{\vec{C}_{\sigma_{w,1}}, \vec{C}_{\sigma_{w,2}}, \vec{\pi}_w\}_{w \in W}, \vec{\pi}_{sum} \end{pmatrix} \in \mathbb{G}^{28+7|M|+9\,W|} \tag{7}$$

`Derive`$(pk, (M, W), \sigma, (M', W'))$**:** Parse σ as (7). Assume that $1 \leftarrow \mathcal{P}_{\texttt{mixed}}((M, W), (M', W'))$. Conduct the following six steps.

1. Re-randomize the GS commitments $\vec{C}_X, \vec{C}_Y \in \mathbb{G}^3$. For each variable $Z \in \{X, Y\}$, $\vec{C}'_Z := \vec{C}_Z \cdot \vec{f}_1^{\,r'_Z} \cdot \vec{f}_2^{\,s'_Z} \cdot \vec{f}_3^{\,t'_Z}$, where $r'_Z, s'_Z, t'_Z \xleftarrow{\text{U}} \mathbb{Z}_p$.
2. Randomize the AHO signature $(\theta_1, \cdots, \theta_7) \in \mathbb{G}^7$ in the same manner as [3,4,6] explained in Appendix 4. Choose $\eta_2, \eta_5, \mu, \nu \xleftarrow{\text{U}} \mathbb{Z}_p$, then

$$\begin{aligned}
&\vec{C}'_{\theta_1} := \vec{C}_{\theta_1} \cdot \vec{f}_1^{\,r'_{\theta_1}} \cdot \vec{f}_2^{\,s'_{\theta_1}} \cdot \vec{f}_3^{\,t'_{\theta_1}}, \\
&\vec{C}'_{\theta_2} := \vec{C}_{\theta_2} \cdot \iota_{\mathbb{G}}(\theta_4^{\eta_2}) \cdot \vec{f}_1^{\,r'_{\theta_2}} \cdot \vec{f}_2^{\,s'_{\theta_2}} \cdot \vec{f}_3^{\,t'_{\theta_2}}, \quad \theta'_3 := (\theta_3 \cdot g_r^{-\eta_2})^{1/\mu}, \quad \theta'_4 := \theta_4^{\mu}, \\
&\vec{C}'_{\theta_5} := \vec{C}_{\theta_5} \cdot \iota_{\mathbb{G}}(\theta_7^{\eta_5}) \cdot \vec{f}_1^{\,r'_{\theta_5}} \cdot \vec{f}_2^{\,s'_{\theta_5}} \cdot \vec{f}_3^{\,t'_{\theta_5}}, \quad \theta'_6 := (\theta_6 \cdot h_r^{-\eta_5})^{1/\nu}, \quad \theta'_7 := \theta_7^{\nu},
\end{aligned}$$

where $r'_{\theta_i}, r'_{\theta_i}, t'_{\theta_i} \xleftarrow{\mathrm{U}} \mathbb{Z}_p$ for each $i \in \{1, 2, 5\}$. The GS proofs $\vec{\pi}_A, \vec{\pi}_B \in \mathbb{G}^3$ are naturally updated as follows.

$$\vec{\pi}'_A := \vec{\pi}_A \cdot (g_z^{r'_{\theta_1}} g_r^{r'_{\theta_2}} g_1^{r'_X} g_2^{r'_Y}, g_z^{s'_{\theta_1}} g_r^{s'_{\theta_2}} g_1^{s'_X} g_2^{s'_Y}, g_z^{t'_{\theta_1}} g_r^{t'_{\theta_2}} g_1^{t'_X} g_2^{t'_Y})$$
$$\vec{\pi}'_B := \vec{\pi}_B \cdot (h_z^{r'_{\theta_1}} h_r^{r'_{\theta_2}} h_1^{r'_X} h_2^{r'_Y}, h_z^{s'_{\theta_1}} h_r^{s'_{\theta_2}} h_1^{s'_X} h_2^{s'_Y}, h_z^{t'_{\theta_1}} h_r^{t'_{\theta_2}} h_1^{t'_X} h_2^{t'_Y})$$

3. For each $m \in M'$, re-randomize the Waters signature $(\sigma_{m,1}, \sigma_{m,2}) \in \mathbb{G}$. Choose $\chi'_m \xleftarrow{\mathrm{U}} \mathbb{Z}_p$, then compute $\sigma'_{m,2} := \sigma_{m,2} \cdot g^{\chi'_m}$ and $\vec{C}'_{\sigma_{m,1}} := \vec{C}_{\sigma_{m,1}} \cdot \iota_{\mathbb{G}}(H_{\mathbb{G}}(m)^{\chi'_m}) \cdot \vec{f}_1^{\,r'_{\sigma_{m,1}}} \cdot \vec{f}_2^{\,s'_{\sigma_{m,1}}} \cdot \vec{f}_3^{\,t'_{\sigma_{m,1}}}$, where $r'_{\sigma_{m,1}}, s'_{\sigma_{m,1}}, t'_{\sigma_{m,1}} \xleftarrow{\mathrm{U}} \mathbb{Z}_p$. The GS proof $\vec{\pi}_m$ is naturally updated to $\vec{\pi}'_m := \vec{\pi}_m \cdot (g^{r'_{\sigma_{m,1}}} \cdot h^{-r'_X}, g^{s'_{\sigma_{m,1}}} \cdot h^{-s'_X}, g^{t'_{\sigma_{m,1}}} \cdot h^{-t'_X})$.
4. Choose $\{\gamma'_w \in \mathbb{Z}_p\}_{w \in W'}$ s.t. $\sum_{w \in W'} \gamma'_w = 0 \pmod p$ uniformly at random from $\mathbb{Z}_p$.
5. For each $w \in W$, re-randomize $(\sigma_{w,1}, \sigma_{w,2}) \in \mathbb{G}^2$ and their commitments $\vec{C}_{\sigma_{w,1}}, \vec{C}_{\sigma_{w,2}} \in \mathbb{G}^3$. Compute $\vec{C}'_{\sigma_{w,1}} := \vec{C}_{\sigma_{w,1}} \cdot \iota_{\mathbb{G}}(H_{\mathbb{G}}(w)^{\gamma'_w}) \cdot \vec{f}_1^{\,r'_{\sigma_{w,1}}} \cdot \vec{f}_2^{\,s'_{\sigma_{w,1}}} \cdot \vec{f}_3^{\,t'_{\sigma_{w,1}}}$ and $\vec{C}'_{\sigma_{w,2}} := \vec{C}_{\sigma_{w,2}} \cdot \iota_{\mathbb{G}}(g^{\gamma'_w}) \cdot \vec{f}_1^{\,r'_{\sigma_{w,2}}} \cdot \vec{f}_2^{\,s'_{\sigma_{w,2}}} \cdot \vec{f}_3^{\,t'_{\sigma_{w,2}}}$, where $r'_{\sigma_{w,1}}, s'_{\sigma_{w,1}}, t'_{\sigma_{w,1}} \xleftarrow{\mathrm{U}} \mathbb{Z}_p$. The GS proof $\vec{\pi}_w$ is naturally updated to $\vec{\pi}'_w := \vec{\pi}_w \cdot (g^{r'_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-r'_{\sigma_{w,2}}}, g^{s'_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-s'_{\sigma_{w,2}}}, g^{t'_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-t'_{\sigma_{w,2}}})$.
6. For each $w \in W' \setminus W$, newly generate $(\sigma_{w,1}, \sigma_{w,2}) := (H_{\mathbb{G}}(w)^{\gamma'_w}, g^{\gamma'_w})$. Then generate a GS commitments $\vec{C}'_{\sigma_{w,1}} := \iota_{\mathbb{G}}(H_{\mathbb{G}}(w)^{\gamma'_w}) \cdot \vec{f}_1^{\,r'_{\sigma_{w,1}}} \cdot \vec{f}_2^{\,s'_{\sigma_{w,1}}} \cdot \vec{f}_3^{\,t'_{\sigma_{w,1}}}$ and $\vec{C}'_{\sigma_{w,2}} := \iota_{\mathbb{G}}(g^{\gamma'_w}) \cdot \vec{f}_1^{\,r'_{\sigma_{w,2}}} \cdot \vec{f}_2^{\,s'_{\sigma_{w,2}}} \cdot \vec{f}_3^{\,t'_{\sigma_{w,2}}}$, where $r'_{\sigma_{w,1}}, s'_{\sigma_{w,1}}, t'_{\sigma_{w,1}}, r'_{\sigma_{w,2}}, s'_{\sigma_{w,2}}, t'_{\sigma_{w,2}} \xleftarrow{\mathrm{U}} \mathbb{Z}_p$. Then generate a GS proof $\vec{\pi}'_w := (g^{r'_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-r'_{\sigma_{w,2}}}, g^{s'_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-s'_{\sigma_{w,2}}}, g^{t'_{\sigma_{w,1}}} \cdot H_{\mathbb{G}}(w)^{-t'_{\sigma_{w,2}}})$.

Finally, output a signature σ' composed of all of the updated variables.

$\mathtt{Ver}(pk, (M, W), \sigma)$: Parse σ as (7). Each GS proof $\pi \in \mathbb{G}^3$ is parsed as (π_1, π_2, π_3). Output 1 if and only if all of the following five equations hold.

1. $\iota_{\mathbb{G}_T}(A) \cdot e(\theta_3, \iota_{\mathbb{G}}(\theta_4))^{-1} = E(g_z, \vec{C}_{\theta_1}) \cdot E(g_r, \vec{C}_{\theta_2}) \cdot E(g_1, \vec{C}_X) \cdot E(g_2, \vec{C}_Y) \cdot \prod_{k=1}^{3} E(\pi_{A,k}, \vec{f}_k)$
2. $\iota_{\mathbb{G}_T}(B) \cdot e(\theta_6, \iota_{\mathbb{G}}(\theta_7))^{-1} = E(h_z, \vec{C}_{\theta_1}) \cdot E(h_r, \vec{C}_{\theta_5}) \cdot E(h_1, \vec{C}_X) \cdot E(h_2, \vec{C}_Y) \cdot \prod_{k=1}^{3} E(\pi_{B,k}, \vec{f}_k)$
3. $E(g, \vec{C}_{\sigma_{m,1}}) = E(h, \vec{C}_X) \cdot E(H_{\mathbb{G}}(m), \iota_{\mathbb{G}}(\sigma_{m,2})) \cdot \prod_{k=1}^{3} E(\pi_{m,k}, \vec{f}_k)$ (for each $m \in M$)
4. $E(g, \vec{C}_{\sigma_{w,1}}) = E(H_{\mathbb{G}}(w), \vec{C}_{\sigma_{w,2}}) \cdot \prod_{k=1}^{3} E(\pi_{w,k}, \vec{f}_k)$ (for each $w \in W$)
5. $E(g, \vec{C}_Y) = \prod_{w \in W} E(g, \vec{C}_{\sigma_{w,2}}) \cdot \prod_{k=1}^{3} E(\pi_{sum,k}, \vec{f}_k)$

Theorem 1. *Our HSSM scheme is* $\mathtt{CCH}$ *unconditionally, and* $\mathtt{wUNF}$ *under the DLIN assumption w.r.t the group* $\mathbb{G}$ *and the q-SFP assumption, where* $q \in \mathtt{poly}(\lambda)$ *is the maximal number of times that the signing oracle can be accessed.*

Proof. Among various variables included in a derived signature, GS commitments and proofs distribute identically to fresh ones because of the following two facts, (1) GS NIWI proof system is perfectly WI, and (2) they are perfectly re-randomized in the derivation algorithm. The other variables, i.e., $\{\theta_i\}_{i\in\{3,4,6,7\}}$ and $\{\sigma_{m,2}\}_{m\in M'}$, also distribute identically to fresh ones because they are perfectly re-randomized in the derivation algorithm. Hence, our HSSM scheme is CCH. To prove its wUNF, we define the following 4 experiments.

Expt$_0$**:** This is the standard wUNF experiment for the HSSM scheme. The forged signature σ^* is parsed as $(\vec{C}^*_X, \vec{C}^*_Y, \{\vec{C}^*_{\theta_i}\}_{i\in\{1,2,5\}}, \{\theta^*_i\}_{i\in\{3,4,6,7\}}, \vec{\pi}^*_A, \vec{\pi}^*_B, \{\vec{C}^*_{\sigma_{m,1}}, \sigma^*_{m,2}, \vec{\pi}^*_m\}_{m\in M^*}, \{\vec{C}^*_{\sigma_{w,1}}, \vec{C}^*_{\sigma_{w,2}}, \vec{\pi}^*_w\}_{w\in W^*}, \vec{\pi}^*_{sum})$.

Expt$_1$**:** The same as ***Expt***$_0$ except that the GS CRS $\boldsymbol{f} = (\vec{f}_1, \vec{f}_2, \vec{f}_3)$ is generated as a perfectly sound one. Specifically, $\vec{f}_1 := (f_1, 1, g)$, $\vec{f}_2 := (1, f_2, g)$ and $\vec{f}_3 := \vec{f}_1^{\xi_1} \cdot \vec{f}_2^{\xi_2}$, where $f_1 := g^{\phi_1}$ and $f_2 := g^{\phi_2}$ with $\phi_1, \phi_2, \xi_1, \xi_2 \xleftarrow{U} \mathbb{Z}_p$. Note that in this and later experiments, all GS commitments are perfectly binding. From the forged GS commitments, we can extract all of the hidden variables $X^*, Y^*, \{\theta^*_i\}_{i\in\{1,2,5\}}, \{\sigma^*_{m,1}\}_{m\in M^*}$ and $\{\sigma^*_{w,1}, \sigma^*_{w,2}\}_{w\in W^*}$ by using the BBS decryption keys (ϕ_1, ϕ_2). Since the forged GS proofs are perfectly sound, the extracted variables satisfy all of the five Eqs. (2)–(6).

Expt$_2$**:** For $\kappa \in [1, q]$, $(X_\kappa, Y_\kappa) \in \mathbb{G}^2$ denote the group elements (X, Y) randomly chosen on the κ-th query to the signing oracle. ***Expt***$_2$ is the same as ***Expt***$_1$ except that it aborts if $\nexists \kappa \in [1, q]$ s.t. $(X_\kappa, Y_\kappa) = (X^*, Y^*)$.

Expt$_3$**:** Identical to ***Expt***$_2$ except that it aborts if $\exists \kappa \in [1, q]$ s.t. $(X_\kappa, Y_\kappa) = (X^*, Y^*) \wedge M^* \nsubseteq M_\kappa$, where M_κ is the κ-th query of M to the signing oracle.

S_i denotes the event where ***Expt***$_i$ outputs 1. We obtain $\mathsf{Adv}^{\mathtt{wUNF}}_{\Sigma_{\mathrm{HSSM}},\mathcal{A},n}(\lambda) = \Pr[S_0] \leq \sum_{i=1}^{3} |\Pr[S_{i-1}] - \Pr[S_i]| + \Pr[S_3]$. Because of the following four lemmas, the rightmost formula is negligible under the DLIN and q-SFP assumptions[2]. Lemma 1 is true as shown in [6,20]. □

Lemma 1. $|\Pr[S_0] - \Pr[S_1]|$ *is negligible under the DLIN assumption w.r.t.* $\mathbb{G}$.

Lemma 2. $|\Pr[S_1] - \Pr[S_2]|$ *is negligible under the q-SFP assumption.*

Proof. Let Abort denote the aborting event added in ***Expt***$_2$. Since ***Expt***$_2$ is the same as ***Expt***$_1$ except for the case that the aborting event occurs, $\Pr[S_2] = \Pr[S_1 \wedge \neg\mathsf{Abort}]$. By a basic mathematical theorem, $\Pr[S_1] - \Pr[S_1 \wedge \neg\mathsf{Abort}] = \Pr[S_1] - \Pr[S_2] = \Pr[S_1 \wedge \mathsf{Abort}]$. Let $\mathcal{A}$ denote a PPT adversary which makes the event $S_1 \wedge \mathsf{Abort}$ occurs with a non-negligible probability. Let $\mathcal{B}_2$ denote a PPT adversary which, by using $\mathcal{A}$ as black-box, attempts to win the existential unforgeability against adaptively chosen messages attacks (EUF-CMA) experiment (defined in Appendix 1) w.r.t. the AHO SPS scheme. $\mathcal{B}_2$ behaves as follows.

$\mathcal{B}_2$ receives an honestly-generated public-key pk_{s} of the AHO scheme. $\mathcal{B}_2$ honestly generates a GS CRS $\boldsymbol{f}$ in the perfect soundness setting and public parameters $h, u', u_0, \cdots, u_{L-1}$ of the Waters signatures. If $\mathcal{A}$ issues (M, W) as a

[2] The CDH assumption is implied by the DLIN and q-SFP assumptions.

query to the signing oracle, $\mathcal{B}_2$ generates $(X, Y) := (g^x, g^y)$ (where $x, y \xleftarrow{\text{U}} \mathbb{Z}_p$), then sends the message (X, Y) as a query to the signing oracle to get an AHO signature σ_{s}. $\mathcal{B}_2$ honestly generates the other elements of the HSSM signature σ on (M, W). Since we have assumed that the event $S_1 \wedge \mathsf{Abort}$ occurs, (1) *σ_{s}^* is a valid AHO signature on the message* (X^*, Y^*), and (2) *the message has not been queried to the signing oracle by $\mathcal{B}_2$*. Thus, $\mathcal{B}_2$ wins. It holds that $\Pr[S_1 \wedge \mathsf{Abort}] \leq \mathtt{Adv}^{\mathtt{EUF\text{-}CMA}}_{\Sigma_{\text{AHO}}, \mathcal{B}_2}(\lambda)$, where the right side denotes $\mathcal{B}_2$'s advantage in the $\mathtt{EUF\text{-}CMA}$ experiment which is negligible under the q-SFP assumption. □

Lemma 3. $|\Pr[S_2] - \Pr[S_3]|$ *is negligible under the CDH assumption w.r.t.* $\mathbb{G}$.

Proof. Let Abort denote the aborting event added in $\boldsymbol{Expt}_3$. As the proof of Lemma 2, it holds that $\Pr[S_2] - \Pr[S_3] = \Pr[S_2 \wedge \mathsf{Abort}]$. Let $\mathcal{A}$ denote a PPT adversary which makes the event $S_2 \wedge \mathsf{Abort}$ occurs with a non-negligible probability. Let $\mathcal{B}_3$ denote a PPT adversary which, by using $\mathcal{A}$ as black-box, attempts to win the $\mathtt{EUF\text{-}CMA}$ experiment w.r.t. the Waters scheme. $\mathcal{B}_3$ behaves as follows.

$\mathcal{B}_3$ receives honestly-generated public parameters $h, u', u_0, \cdots, u_{L-1}$ and a public-key $X'(:= g^{x'})$ (where $x' \xleftarrow{\text{U}} \mathbb{Z}_p$) of the Waters signature. $\mathcal{B}_3$ honestly generates a GS CRS $\boldsymbol{f}$ in the perfect soundness setting and an AHO key-pair $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$. Since we have assumed that the event $S_2 \wedge \mathsf{Abort}$ occurs, it will hold that $\exists \kappa \in [1, q]$ s.t. $(X_\kappa, Y_\kappa) = (X^*, Y^*) \wedge M_\kappa \not\supseteq M^*$. $\mathcal{B}_3$ guesses such an index κ and chooses $\kappa_{guess} \xleftarrow{\text{U}} [1, q]$. The guess is correct at least with probability $1/q$. $\mathcal{B}_3$ proceeds under an assumption that the guess is correct. If $\mathcal{A}$ issues (M, W) as the κ-th query to the signing oracle, $\mathcal{B}_3$ considers the following two cases.

1. $\kappa \neq \kappa_{guess}$**:** $\mathcal{B}_3$ honestly generates the whole HSSM signature σ oneself.
2. $\kappa = \kappa_{guess}$**:** $\mathcal{B}_3$ uses the given $X' \in \mathbb{G}$ as the Waters public-key. For each $m \in M$, $\mathcal{B}_3$ queries $m \in \{0, 1\}^L$ to the signing oracle to get a Waters signature $(\sigma_{m,1}, \sigma_{m,2})$. $\mathcal{B}_3$ honestly generates the HSSM signature σ oneself.

Since we have assumed that $S_2 \wedge \mathsf{Abort}$ occurs, (1) *there exists $m^* \in M^*$ s.t. $m^* \notin M_{\kappa_{guess}}$*, (2) *the Waters signature $(\sigma^*_{m^*,1}, \sigma^*_{m^*,2})$ on m^* is valid*, and (3) *the message m^* has not been queried to the signing oracle by $\mathcal{B}_3$*. Thus, $\mathcal{B}_3$ wins (under the assumption that $\mathcal{B}_3$'s guess κ_{guess} is correct). It holds that $\Pr[S_2 \wedge \mathsf{Abort}] \leq q \cdot \mathtt{Adv}^{\mathtt{EUF\text{-}CMA}}_{\Sigma_{\text{Waters}}, \mathcal{B}_3}(\lambda)$, where $\mathtt{Adv}^{\mathtt{EUF\text{-}CMA}}_{\Sigma_{\text{Waters}}, \mathcal{B}_3}(\lambda)$ is $\mathcal{B}_3$'s advantage in the $\mathtt{EUF\text{-}CMA}$ experiment w.r.t. the Waters scheme which is negligible under the CDH assumption. In Appendix 6, we rigorously prove that there exists a PPT adversary $\mathcal{B}_3'$ s.t. $\Pr[S_2 \wedge \mathsf{Abort}] \leq 4q \cdot d_M \cdot (L + 1) \cdot \mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}_3', \mathbb{G}}(\lambda)$, where $d_M \in \mathtt{poly}(\lambda)$ is the maximal cardinality of the set M. □

Lemma 4. $\Pr[S_3]$ *is negligible under the CDH assumption w.r.t.* $\mathbb{G}$.

Proof. We adopt the same proof approach as [20]. We prove that there exists a PPT adversary $\mathcal{B}_4$ s.t. $\Pr[S_3] \leq 4q \cdot d_W \cdot (L+1) \cdot \mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}_4, \mathbb{G}}(\lambda)$, where $d_W \in \mathtt{poly}(\lambda)$ is the maximal cardinality of the set W. Let $\mathcal{A}$ denote a PPT adversary which makes the event S_3 occur with a non-negligible probability. Let $\mathcal{B}_4$ denote a PPT adversary which uses $\mathcal{A}$ to solve the CDH problem. $\mathcal{B}_4$ behaves as follows.

Receive (g, g^a, g^b) as an instance of the CDH problem. Honestly generate a key-pair $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$ of the AHO scheme and a GS CRS $\boldsymbol{f} = (\vec{f}_1, \vec{f}_2, \vec{f}_3)$ in the perfect soundness setting. Then conduct the following two steps.

1. Set $l := 2d_W$. Choose uniformly at random an integer k satisfying $0 \leq k \leq L$. Assume that $l(L+1) \leq p$.
2. Let $h \overset{\text{U}}{\longleftarrow} \mathbb{G}$. Choose $x', x_0, \cdots, x_{L-1} \overset{\text{U}}{\longleftarrow} \mathbb{Z}_l$ and $y', y_0, \cdots, y_{L-1} \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$. For an element $w \in \{0,1\}^L$, define two functions $J, K : \{0,1\}^L \to \mathbb{Z}_p$ as $J(w) := x' + \sum_{i=0}^{L-1} x_i \cdot w[i] - lk$ and $K(w) := y' + \sum_{i=0}^{L-1} y_i \cdot w[i]$. Set $u' := (g^a)^{-lk+x'} \cdot g^{y'}$ and $u_i := (g^a)^{x_i} \cdot g^{y_i}$ for $i \in [0, L-1]$. It holds that $u' \prod_{i=0}^{L-1} u_i^{w[i]} = (g^a)^{-lk+x'+\sum_{i=0}^{L-1} x_i \cdot w[i]} \cdot g^{y'+\sum_{i=0}^{L-1} y_i \cdot w[i]} = (g^b)^{J(w)} \cdot g^{K(w)}$.

Set $pk := (\mathbb{G}, \mathbb{G}_T, g, h, u', \{u_i\}_{i=0}^{L-1}, pk_{\mathsf{s}}, \boldsymbol{f})$ and send it to $\mathcal{A}$. Since we have assumed that the event S_3 occurs, it must hold that $\exists \kappa \in [1, q]$ s.t. $(X_\kappa, Y_\kappa) = (X^*, Y^*) \wedge W_\kappa \nsubseteq W^*$. $\mathcal{B}_4$ guesses such an index κ by $\kappa_{guess} \overset{\text{U}}{\longleftarrow} [1, q]$. The guess is correct at least with probability $1/q$. $\mathcal{B}_4$ proceeds under an assumption that the guess is correct. When $\mathcal{A}$ queries to the signing oracle, $\mathcal{B}_4$ behaves as follows.

$\mathfrak{Sign}(M, W)$**:** Assume that this query is the κ-th query to the oracle. $\mathcal{B}_4$ considers the following two cases.

1. $\kappa \neq \kappa_{guess}$**:** $\mathcal{B}_4$ honestly generates the whole HSSM signature σ oneself, then returns it to $\mathcal{A}$.

2. $\kappa = \kappa_{guess}$**:** If $\nexists w \in W$ s.t. $J(w) = 0 \pmod p$, $\mathcal{B}_4$ aborts the simulation. Otherwise, $\exists w \in W$ s.t. $J(w) = 0 \pmod p$. Hereafter, such an element w is denoted by w'. $\mathcal{B}_4$ sets $Y := g^b$. For each $w \in W \setminus \{w'\}$, $\mathcal{B}_4$ chooses $\gamma_w \overset{\text{U}}{\longleftarrow} \mathbb{Z}_p$ then computes $(\sigma_{w,1}, \sigma_{w,1}) := (H_{\mathbb{G}}(w)^{\gamma_w}, g^{\gamma_w})$. $\mathcal{B}_4$ computes $(\sigma_{w',1}, \sigma_{w',2}) := ((g^b \cdot g^{-\sum_{w \in W \setminus \{w'\}} \gamma_w})^{K(w')}, g^b \cdot g^{-\sum_{w \in W \setminus \{w'\}} \gamma_w})$. Since $J(w') = 0 \pmod p$, they distribute as $(H_{\mathbb{G}}(w')^{\gamma_{w'}}, g^{\gamma_{w'}})$, where $\gamma_{w'} := b - \sum_{w \in W \setminus \{w'\}} \gamma_w$. $\mathcal{B}_4$ honestly generates the other elements of the HSSM signature σ oneself, then returns σ to $\mathcal{A}$.

$\mathcal{B}_4$ receives a forged signature σ^* sent by $\mathcal{A}$. Since we have assumed that the event S_3 occurs, all of the following three conditions hold, namely (a) $Y^* = Y_{\kappa_{guess}}$, (b) $W^* \nsupseteq W_{\kappa_{guess}}$, and (c) *there exist* $\{\gamma_w \in \mathbb{Z}_p\}_{w \in W^*}$ *s.t.* $(\sigma^*_{w,1}, \sigma^*_{w,2}) = (H_{\mathbb{G}}(w)^{\gamma_w}, g^{\gamma_w})$ *for each* $w \in W^*$ *and* $\sum_{w \in W^*} \gamma_w = b$. $\mathcal{B}_4$ aborts the simulation if $\exists w \in W^*$ s.t. $J(w) = 0 \pmod l$. Otherwise, for each $w \in W^*$, $J(w) \neq 0 \pmod l$, which implies $J(w) \neq 0 \pmod p$. $\mathcal{B}_4$ computes $\prod_{w \in W^*} \left\{ \frac{\sigma^*_{w,1}}{(\sigma^*_{w,2})^{K(w)}} \right\}^{\frac{1}{J(w)}} = \prod_{w \in W^*} \left\{ \frac{(g^a)^{J(w) \cdot \gamma_w} \cdot g^{K(w) \cdot \gamma_w}}{g^{\gamma_w \cdot K(w)}} \right\}^{\frac{1}{J(w)}} = \prod_{w \in W^*} (g^a)^{\gamma_w} = (g^a)^{\sum_{w \in W^*} \gamma_w} = g^{ab}$ then outputs it as the answer to the CDH problem.

Let $\mathsf{SimAbort}$ denote the event that $\mathcal{B}_4$ aborts the simulation. At least when S_3 has occurred and $\mathcal{B}_4$ has not aborted the simulation, $\mathcal{B}_4$ finds the correct answer to the CDH problem. Thus, it holds $\mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}_4, \mathbb{G}}(\lambda) \geq \Pr[S_3 \wedge \neg \mathsf{SimAbort}] = \Pr[\neg \mathsf{SimAbort} \mid S_3] \cdot \Pr[S_3]$, implying $\Pr[S_3] \leq \frac{1}{\Pr[\neg \mathsf{SimAbort} \mid S_3]} \cdot \mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}_4, \mathbb{G}}(\lambda)$.

We analyze the probability $\Pr[\neg \mathsf{SimAbort} \mid S_3]$. We define three events.

E_1: $(X^*, Y^*) = (X_{\kappa_{guess}}, Y_{\kappa_{guess}})$
E_2: $\exists w' \in W_{\kappa_{guess}}$ s.t. $w' \notin W^* \wedge J(w') = 0 \pmod{p}$
E_3: $\forall w \in W^*$, $J(w) \neq 0 \pmod{l}$

We obtain

$$\begin{aligned}
&\Pr[\neg\mathsf{SimAbort} \mid S_3] \\
&= \Pr[\mathsf{E}_1 \wedge \mathsf{E}_2 \wedge \mathsf{E}_3 \mid S_3] \\
&= \Pr[\mathsf{E}_1 \mid S_3] \cdot \Pr[\mathsf{E}_2 \wedge \mathsf{E}_3 \mid S_3 \wedge \mathsf{E}_1] \\
&= \Pr[\mathsf{E}_1 \mid S_3] \cdot \Pr[\mathsf{E}_2 \mid S_3 \wedge \mathsf{E}_1] \cdot \Pr[\mathsf{E}_3 \mid S_3 \wedge \mathsf{E}_1 \wedge \mathsf{E}_2].
\end{aligned}$$

We analyze each term. Obviously, $\Pr[\mathsf{E}_1 \mid S_3] \geq 1/q$. The second term is analyzed as follows.

$$\begin{aligned}
&\Pr[\mathsf{E}_2 \mid S_3 \wedge \mathsf{E}_1] \\
&= \Pr[J(w') = 0 \pmod{p} \mid S_3 \wedge \mathsf{E}_1] \\
&= \Pr[J(w') = 0 \pmod{l} \mid S_3 \wedge \mathsf{E}_1] \\
&\quad \cdot \Pr[J(w') = 0 \pmod{p} \mid S_3 \wedge \mathsf{E}_1 \wedge J(w') = 0 \pmod{l}] \\
&= \frac{1}{l}\frac{1}{L+1}
\end{aligned}$$

The third term is as follows.

$$\begin{aligned}
&\Pr[\mathsf{E}_3 \mid S_3 \wedge \mathsf{E}_1 \wedge \mathsf{E}_2] \\
&= \Pr\left[\bigwedge_{w \in W^*} J(w) \neq 0 \pmod{l} \middle| S_3 \wedge \mathsf{E}_1 \wedge \mathsf{E}_2\right] \\
&\geq 1 - \sum_{w \in W^*} \Pr[J(w) = 0 \pmod{l} \mid S_3 \wedge \mathsf{E}_1 \wedge \mathsf{E}_2] \\
&= 1 - \sum_{w \in W^*} \frac{1}{l} \geq 1 - \frac{d_W}{l}
\end{aligned}$$

As a result, we obtain

$$\Pr[\neg\mathsf{SimAbort} \mid S_3] \geq \frac{1}{q} \cdot \frac{1}{l} \cdot \frac{1}{L+1} \cdot \left(1 - \frac{d_W}{l}\right) = \frac{1}{4q \cdot d_W \cdot (L+1)}$$

Therefore, $\Pr[S_3] \leq 4q \cdot d_W \cdot (L+1) \cdot \mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}_4,\mathbb{G}}(\lambda)$. □

4.2 Another Construction from the DLIN Assumption Only

By replacing the AHO SPS scheme [3,4] in the above HSSM scheme with Abe et al.'s SPS scheme [2] satisfying unforgeability against extended random messages attacks (UF-XRMA) (defined in Subsect. Appendix 1) under the DLIN assumption, we obtain an HSSM scheme secure under the DLIN assumption only. In the

Table 1. Comparison among existing HSSB/RS schemes. CH and DC mean context-hiding and disclosure-controllability, respectively.

HSSB/RS Schemes	CH	DC	Assumption
ABC+12 [5] w. [18]	Strong	–	Subgroup Decision [19]
ALP12 [6]	Complete	–	DLIN
Ours	Complete	✓	DLIN

modified HSSM scheme, the signer chooses $x, y \xleftarrow{\text{U}} \mathbb{Z}_p$, then generates an Abe et al.'s SPS signature $(\theta_0, \cdots, \theta_7) \in \mathbb{G}^8$ on a message composed of six group elements $(M_1, \cdots, M_6) := (C^x, C^y, F^x, F^y, U_1^x, U_2^y) \in \mathbb{G}^6$, where $U, F, U_1, U_2 \in \mathbb{G}$ are group generators. For each element in $\{M_1, \cdots, M_6, \theta_0, \cdots, \theta_7\}$, the signer computes a GS commitment. The verification algorithm of the Abe et al.'s SPS scheme consists of seven PPEs. For each PPE, the signer computes a GS proof.

5 Applications

5.1 Disclosure-Controllable (DC) HSSB

In HSSB [5,6], any signature on a set M generates a signature on any subset $M' \subseteq M$. In the ordinary HSSB, any sub-message $m \in M$ can be deleted anytime. We define DCHSSB that any deletable sub-message $m \in M$ can be *undeletable* anytime. The change of deletability is one-way, which means that any undeletable sub-message cannot be made deletable again. If every sub-message is undeletable, the message is finalized.

DCHSSB is defined as follows. Given a set M, $D_M (\subseteq M)$ denotes the set of its deletable sub-messages. From a signature on M with D_M, we can derive a signature on M' with a set $D'_{M'}$ of its deletable sub-messages, if $M \subseteq M'$ and $D_M \supseteq D'_{M'}$. Obviously, DCHSSB is a subclass of HSSM. Specifically, DCHSSB is identical to HSSM with a restriction that *any message* (M, W) *satisfies* $M \supseteq W$.

From our HSSM scheme in Subsect. 4.2, a DCHSSB scheme secure under the DLIN assumption is derived. To the best of our knowledge, there has been two HSSB schemes which satisfy all of the three conditions, namely **C1:** adaptively unforgeable[3], **C2:** strongly context-hiding, and **C3:** secure under standard assumptions. They are the scheme by Attrapadung et al. [6] secure under the DLIN assumption, and the scheme by Ahn et al. [5] instantiated with the ciphertext-policy attribute-based encryption (CP-ABE) scheme [18]. Since their disclosure-controllability have not been proven, our scheme is the first disclosure-controllable one satisfying all of the above three conditions. See Table 1.

[3] Our unforgeability with Definition 4 is adaptive. In selective unforgeability [5], the adversary $\mathcal{A}$ must choose the target message M^* before receiving the public-key pk.

5.2 Disclosure-Controllable Redactable Signatures (DCRS)

In redactable signatures (RS) with maximal number of sub-messages $N \in \mathbb{N}$, each message M is an ordered list in the form of $(m_1, \cdots, m_n)$ for some $n \in [1, N]$, where $m_i \in \{0,1\}^L \cup \{*\}$. Each (non-redacted) sub-message $m_i (\neq *)$ can be changed to $*$, which means it has been redacted, i.e., blacked out. RS is a subclass of $\mathcal{P}$-HS defined in Sect. 3. The predicate $\mathcal{P}_{\mathtt{redact}}$ takes M and M', then outputs 1 iff $n = n' \bigwedge_{i=1}^{n} m_i \neq m'_i \implies m'_i = *$, where M' is parsed as $(m'_1, \cdots, m'_{n'})$ for some $n' \in [1, N]$. A RS scheme (parameterized by L and N) can be transformed from an HSSB scheme with sub-message length $L' := L + 1 + \log(N + 1)$ and maximal cardinality of message $K := N + 1$. A RS message $M = (m_1, \cdots, m_n)$ is changed to an HSSB message $M' = \bigcup_{i=1}^{n} \{i \parallel 0 \parallel m_i\} \bigcup \{n + 1 \parallel 1 \parallel 1^L\}$. Redacting m_i in M is deleting the element $i \parallel 0 \parallel m_i$ from M'[4].

In the ordinary RS, any (non-redacted) sub-message can be redacted anytime. In DCRS, any sub-message which is *non-redacted* and *redactable* can be *unredactable*. Specifically, each sub-message has one of the following three states, namely **S1**: not redacted yet and redactable, **S2**: already redacted, and **S3**: not redacted yet and unredactable. Any state only transitions from **S1** to **S2** or from **S1** to **S3**. If every sub-message is in **S2** or **S3**, the message is finalized.

DCRS is defined as follows. Each message $M = (m_1, \cdots, m_n)$ is paired with $R_M \subseteq [1, n]$ which is a set of indices for redactable sub-messages in M. The predicate $\mathcal{P}_{\mathtt{dc\text{-}redact}}$ takes (M, R_M) and $(M', R'_{M'})$, then outputs 1 iff $n = n' \bigwedge R_M \subseteq R'_{M'} \bigwedge_{i=1}^{n} m_i \neq m'_i \implies m'_i = * \wedge i \in R_M$. Any DCHSSB scheme can be transformed into a DCRS scheme basically in the same manner as the above transformation from HSSB to RS. A RS message $M = (m_1, \cdots, m_n)$ is changed to the above HSSB message M'. For any $i \in R_M$, the element $i \parallel 0 \parallel m_i \in M'$ is designated as an undeletable sub-message.

From our DCHSSB scheme, a DCRS scheme secure under the DLIN assumption is derived. By applying the Attrapadung et al.'s HSSB scheme [5] and the Ahn et al.'s HSSB scheme [5] instantiated with [18] to the above HSSB-to-RS transformation, we obtain secure RS schemes. Because they are not DC, ours is the first DCRS satisfying the conditions **C1**, **C2** and **C3**. See Table 1.

5.3 Efficient Superset Predicate Signatures (SPPS)

SBPS is the digital signature analogue of subset predicate encryption [16]. SPPS is the *superset* analogue of SBPS. We consider a stronger primitive in a sense that it has *key-delegatability*. SPPS is a subclass of the following key-delegatable predicate signatures (KDPS) which is formally defined in Subsect. Appendix 5.

In KDPS, setup algorithm `Setup`, given a security parameter 1^λ, generates a public-parameter pp and master-key mk. Key-generation `KGen` generates a secret-key for a key index $X \in \mathbf{X}$. Key-delegation `KDel`, given a secret-key for $X \in \mathbf{X}$,

[4] RS with fixed number of sub-messages $N \in \mathbb{N}$ can be obtained in a simpler way. A RS message $M = (m_1, \cdots, m_N)$ is changed to an HSSB message $M' = \bigcup_{i=1}^{n} \{i \| m_i\}$. Redacting the sub-message m_i in M is just deleting the element $i \parallel m_i$ from M'.

generates a secret-key for a key-index $X' \in \mathbf{X}$ s.t. a key predicate $\mathcal{P}_{\mathsf{X}} : \mathbf{X}^2 \rightarrow \{0,1\}$ holds between X and X', i.e., $1 \leftarrow \mathcal{P}_{\mathsf{X}}(X, X')$. Signing algorithm $\mathtt{Sig}$, given a secret-key for $X \in \mathbf{X}$, generates a signature on a message $M \in \mathcal{M}$ associated with a signature index $Y \in \mathbf{Y}$ s.t. a signature predicate $\mathcal{P}_{\mathsf{Y}} : \mathbf{X} \times \mathbf{Y} \rightarrow \{0,1\}$ holds, i.e., $1 \leftarrow \mathcal{P}_{\mathsf{Y}}(X, Y)$. Verification $\mathtt{Ver}$ verifies a signature. As security for KDPS, we require unforgeability and signer-privacy.

As a concrete notion of unforgeability, we define (weak) existential unforgeability against adaptively-chosen messages and predicate attacks, abbreviated as $\mathtt{EUF\text{-}CMA}$. We define an experiment, where a PPT adversary $\mathcal{A}$ receives an honestly-generated public-parameter pp then adaptively accesses two oracles, namely (key-)revelation and signing oracles. The former, given a key index $X \in \mathbf{X}$, returns a secret-key for X. The latter, given message $M \in \mathcal{M}$ and signature index $Y \in \mathbf{Y}$, returns a signature on M associated with Y. $\mathcal{A}$ wins the experiment if $\mathcal{A}$ succeeds in forging a correct signature σ^* on a message $M^* \in \mathcal{M}$ with $Y^* \in \mathbf{Y}$ satisfying both of the two conditions: (1) *Every* $X \in \mathbf{X}$ *queried to the (key-)revelation oracle satisfies* $0 \leftarrow \mathcal{P}_{\mathsf{Y}}(X, Y^*)$ and (2) *Every (*$M \in \mathcal{M}$, $Y \in \mathbf{Y}$*) queried to the signing oracle satisfies* $(M, Y) \neq (M^*, Y^*)$.

Signer-privacy guarantees that any signature reveals no information about the signer's key index X except for the fact that it satisfies the signature index Y. As a notion of signer-privacy, we define perfect signer-privacy. We define two experiments. In the real experiment, a probabilistic algorithm $\mathcal{A}$ queries an honestly-generated secret-key sk for a key index X, a message M and a signature index Y s.t. $1 \leftarrow \mathcal{P}_{\mathsf{Y}}(X, Y)$ to a signing oracle, then receives an honestly-generated signature σ. In the simulated experiment, $\mathcal{A}$ receives a signature σ which has been generated with no information about X or sk. If both experiments are hard to distinguish, the signer-privacy is satisfied.

SPPS is a subclass of KDPS. The message space is $\mathcal{M} := \{0,1\}^N$ for some $N \in \mathtt{poly}(\lambda)$. The key index space and signature index space are $\mathbf{X} := \mathbf{Y} := 2^{\{0,1\}^L}$ for some $L \in \mathtt{poly}(\lambda)$. The key predicate $\mathcal{P}_{\mathsf{X}}$, given $X, X' \in \mathbf{X}$, outputs 1 if $X' \subseteq X$ or 0 otherwise. The signature predicate $\mathcal{P}_{\mathsf{Y}}$, given $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$, outputs 1 if $Y \subseteq X$ or 0 otherwise. A SPPS scheme is obtained from an HSSM scheme with sub-message length $\max\{L, N\} + 1$ as follows.

$\mathtt{Setup}(1^\lambda, L, N)$**:** It generates a key-pair of the HSSM scheme, i.e., $(pp, mk) \leftarrow \text{HSSM}.\mathtt{KGen}(1^\lambda, \max\{L, N\} + 1)$, then outputs it.

$\mathtt{KGen}(mk, X)$**:** It generates an HSSM *signature* sk on a message

$$(M_X, W_X) := \left(\bigcup_{x \in X} \{0 \parallel x\}, \emptyset \right), \tag{8}$$

i.e., $sk \leftarrow \text{HSSM}.\mathtt{Sig}(mk, (M_X, W_X))$, then outputs it.

$\mathtt{KDel}(sk, X')$**:** sk is assumed to be a secret-key for $X \in \mathbf{X}$. Given an HSSM signature sk on (M_X, W_X) in (8), it derives an HSSM signature sk' on $(M_{X'}, W_{X'}) := (\cup_{x \in X'} \{0 \parallel x'\}, \emptyset)$, i.e., $sk' \leftarrow \text{HSSM}.\mathtt{Derive}(pp, (M_X, W_X), sk, (M_{X'}, W_{X'}))$, then outputs it.

$\mathtt{Sig}(sk, Y, m)$**:** sk is assumed to be a secret-key for $X \in \mathbf{X}$. Given an HSSM signature sk on (M_X, W_X) in (8), it derives an HSSM signature σ on

$$(M_Y, W_Y) := \left(\bigcup_{y \in Y} \{0 \parallel y\}, \bigcup_{y \in Y} \{0 \parallel y\} \bigcup \{1 \parallel m\} \right), \tag{9}$$

i.e., $\sigma \leftarrow \text{HSSM}.\mathtt{Derive}(pp, (M_X, W_X), sk, (M_Y, W_Y))$, then outputs it.

$\mathtt{Ver}(\sigma, Y, m)$**:** It verifies the HSSM signature σ on (M_Y, W_Y) in (9). It outputs $1/0 \leftarrow \text{HSSM}.\mathtt{Ver}(pp, (M_Y, W_Y), \sigma)$.

Theorem 2. *The SPPS scheme is* `PRV` *(resp.* `EUF-CMA`*) if the HSSM scheme is* `PRV` *(resp.* `SCH` *and* `wUNF`*).*

Proof. We firstly prove `PRV` then `EUF-CMA`.

The proof of `PRV` is simple. The signing oracle takes a SPPS secret-key sk for X which is an honestly-generated HSSM signature on (M_X, W_X) in (8), then honestly generates a SPPS signature σ which is an HSSM signature on (M_Y, W_Y) in (9). If the HSSM scheme is `SCH`, σ distributes like a fresh signature directly generated by the HSSM secret-key and has no information about X.

For the proof of `EUF-CMA`, we define two experiments. $\boldsymbol{Expt}_0$ is the standard `EUF-CMA` experiment w.r.t. the SPPS scheme. $\boldsymbol{Expt}_1$ is the same as $\boldsymbol{Expt}_0$ except that on the signing oracle the signature σ is directly generated by the HSSM secret-key mk. Specifically, the signing oracle, given $Y \in \mathbf{Y}$ and $m \in \mathcal{M}$, returns $\sigma \leftarrow \text{HSSM}.\mathtt{Sig}(mk, (M_Y, W_Y))$ with (M_Y, W_Y) in (9). If the HSSM scheme is `SCH`, $\boldsymbol{Expt}_1$ distributes identically to $\boldsymbol{Expt}_0$. If the HSSM scheme is `wUNF`, any PPT adversary $\mathcal{A}$ wins the experiment $\boldsymbol{Expt}_1$ only with a negligible probability.

A reduction algorithm $\mathcal{B}$ receives a public-key pp of the HSSM scheme, then gives it to $\mathcal{A}$. When $\mathcal{A}$ queries $(Y \in \mathbf{Y}, m \in \mathcal{M})$ to the signing oracle, $\mathcal{B}$ makes the signing oracle generate an HSSM signature σ on (M_Y, W_Y) in (9), then makes the signature-revelation oracle reveal it. When $\mathcal{A}$ queries $X \in \mathbf{X}$ to the key-revelation oracle, $\mathcal{B}$ makes the signing oracle generate an HSSM signature sk on (M_X, W_X) in (8), then makes the signature-revelation oracle reveal it. Consider a situation where $\mathcal{A}$ wins. For the forged SPPS signature σ^* on m^* associated with $Y^* \in \mathbf{Y}$, following three statements are true. Firstly, σ^* is a correct HSSM signature on $(M_{Y^*}, W_{Y^*}) := (\bigcup_{y \in Y^*} \{0 \parallel y\}, \bigcup_{y \in Y^*} \{0 \parallel y\} \bigcup \{1 \parallel m^*\})$. Secondly, for every X queried to the key-revelation oracle, it holds that $Y^* \not\subseteq X$, which implies that $0 \leftarrow \mathcal{P}_{\mathtt{mixed}}((M_X, W_X), (M_{Y^*}, W_{Y^*}))$. Thirdly, for every (Y, m) queried to the signing oracle, it holds that $(Y, m) \neq (Y^*, m^*)$, which implies that $0 \leftarrow \mathcal{P}_{\mathtt{mixed}}((M_Y, W_Y), (M_{Y^*}, W_{Y^*}))$. Thus, if $\mathcal{A}$ wins, then $\mathcal{B}$ also wins. □

Let us instantiate it by our HSSM scheme secure under the q-SFP and DLIN assumptions in Subsect. 4.1. Since its secret-key for X is an HSSM signature on (M_X, W_X) in (8) with $|M_X| = |X|$ and $|W_X| = 0$, it consists of $28 + 7|M_X| + 9|W_X| = 28 + 7|X|$ number of elements in $\mathbb{G}$. Since its signature for Y is an HSSM signature on (M_Y, W_Y) in (9) with $|M_Y| = |Y|$ and $|W_Y| = |Y| + 1$, it consists of $28 + 7|Y| + 9(|Y| + 1) = 37 + 16|Y|$ group elements.

In fact, SPPS can be obtained from HSSB in an inefficient and somewhat complicated manner. K denotes the maximal cardinality of X or Y. An HSSB with sub-message length $2+\max\{L, 1+\log N, \log K\}$ is needed. A SPPS secret-key for X is an HSSB signature on a set $M_X := \bigcup_{x\in X}\{00 \parallel x\}\bigcup_{i=0}^{|X|-1}\{01 \parallel i\}\bigcup_{j=0}^{N-1}\{10 \parallel j \parallel 0\}\bigcup_{j=0}^{N-1}\{10 \parallel j \parallel 1\}$. A SPPS signature on a message $m \in \{0,1\}^N$ associated with Y is an HSSB signature on $M_Y := \bigcup_{y\in Y}\{00 \parallel y\}\bigcup\{01 \parallel |Y|-1\}\bigcup_{j\in[0,N-1]}\{10 \parallel j \parallel m[j]\}$. We instantiate it by our HSSM scheme in Subsect. 4.1. Its secret-key consists of $28+7(2|X|+2N)+9\cdot 0 = 28+14(|X|+N)$ group elements. Its signature consists of $28+7(|Y|+1+N)+9\cdot 0 = 35+7(|Y|+N)$ group elements. Thus, its secret-key and signature lengths increase linearly with N. To the best of our knowledge, since any existing SCH-secure HSSB scheme has a property that its length of a signature on a set increases linearly with the cardinality of the set, any one of them leads to a SPPS scheme with a property that its secret-key and signature lengths increase linearly with N.

5.4 Efficient Wildcarded Identity-Based Signatures (WIBS)

WIBS is the digital signatures analogue of wildcarded identity-based encryption [1]. WIBS is a subclass of the ordinary (i.e., non key-delegatable) PS. The message space is $\mathcal{M} := \{0,1\}^N$ for some $N \in \mathtt{poly}(\lambda)$. The key index space is $\mathbf{X} := (\{0,1\}^L)^n$ for some $L, n \in \mathtt{poly}(\lambda)$. The signature index space is $\mathbf{Y} := (\{0,1\}^L \cup \{*\})^n$. The signature predicate $\mathcal{P}_\mathsf{Y}$, given $X = (x_1, \cdots, x_n) \in \mathbf{X}$ and $Y = (y_1, \cdots, y_n) \in \mathbf{Y}$, outputs 1 if $y_i \neq * \implies x_i = y_i$ for all $i \in [1,n]$, or 0 otherwise.

An HSSM scheme can be transformed into a WIBS scheme as follows.

- $\mathtt{Setup}(1^\lambda, L, N)$**:** It generates a key-pair of the HSSM scheme with sub-message length $\max\{L+\log L, N\}+1$, i.e., $(pp, mk) \leftarrow \text{HSSM}.\mathtt{KGen}(1^\lambda, \max\{L+\log L, N\}+1)$, then outputs it.
- $\mathtt{KGen}(mk, X)$**:** It generates an HSSM signature sk on a message $(M_X, W_X) := (\bigcup_{i=1}^{n}\{0 \parallel i \parallel x_i\}, \emptyset)$, i.e., $sk \leftarrow \text{HSSM}.\mathtt{Sig}(mk, (M_X, W_X))$, then outputs it.
- $\mathtt{Sig}(sk, Y, m)$**:** Assume that sk is a secret-key for $X \in \mathbf{X}$. Given an HSSM signature sk on (M_X, W_X), it derives an HSSM signature σ on $(M_Y, W_Y) := (\bigcup_{i\in[1,n] \text{ s.t. } y_i\neq *}\{0 \parallel i \parallel y_i\}, \bigcup_{i\in[1,n] \text{ s.t. } y_i\neq *}\{0 \parallel i \parallel y_i\}\bigcup\{1 \parallel m\})$, i.e., $\sigma \leftarrow \text{HSSM}.\mathtt{Derive}(pp, (M_X, W_X), sk, (M_Y, W_Y))$, then outputs it.
- $\mathtt{Ver}(\sigma, Y, m)$**:** It verifies the HSSM signature σ on (M_Y, W_Y). It outputs $1/0 \leftarrow \text{HSSM}.\mathtt{Ver}(pp, (M_Y, W_Y), \sigma)$.

Its security is guaranteed by a theorem similar to Theorem 2. As SPPS, WIBS can also be transformed from HSSB inefficiently.

5.5 Subset Predicate Signatures (SBPS)

SBPS is a subclass of KDPS. The spaces $\mathcal{M}$, $\mathbf{X}$ and $\mathbf{Y}$ are defined as SPPS. $\mathcal{P}_\mathsf{X}$ takes $X, X' \in \mathbf{X}$ then outputs 1 iff $X \subseteq X'$. $\mathcal{P}_\mathsf{Y}$ takes $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$ then outputs 1 iff $X \subseteq Y$.

Identity-based ring signatures (IBRS) [24] is a subclass of SBPS. Specifically, IBRS is identical to SBPS with the following restrictions, namely (1) there is no key-delegation and (2) cardinality of the set $X \in \mathbf{X}$ is fixed to 1. As applications of HSSP, Libert et al. [20] have mentioned only IBRS and append-only signatures [17]. In fact, SBPS can also be considered as an application of HSSP.

SBPS is transformed from HSSP as follows. By $K \in \mathtt{poly}(\lambda)$, we denote the maximal cardinality of the set $X \in \mathbf{X}$ or $Y \in \mathbf{Y}$.

$\mathtt{Setup}(1^\lambda, L, N)$**:** Generate a key-pair of the HSSP scheme with sub-message length $\max\{L, N + \log K\} + 1$, i.e., $(pp, mk) \leftarrow \text{HSSP}.\mathtt{KGen}(1^\lambda, \max\{L, N + \log K\} + 1)$, then output it.

$\mathtt{KGen}(mk, X)$**:** Output $sk \leftarrow \text{HSSP}.\mathtt{Sig}(mk, W_X)$, where $W_X := \bigcup_{x \in X}\{0 \parallel x\}$.

$\mathtt{KDel}(sk, X')$**:** Assume that sk is a for $X \in \mathbf{X}$. Output $sk' \leftarrow \text{HSSP}.\mathtt{Derive}(pp, W_X, sk, W_{X'})$, where $W_{X'} := \cup_{x \in X'}\{0 \parallel x'\}$.

$\mathtt{Sig}(sk, Y, m)$**:** Assume that sk is for $X \in \mathbf{X}$. Cardinality of the set Y is denoted by $|Y| \in [1, K]$. Output $\sigma \leftarrow \text{HSSP}.\mathtt{Derive}(pp, W_X, sk, W_Y)$, where $W_Y := \bigcup_{y \in Y}\{0 \parallel y\} \bigcup \{1 \parallel m \parallel |Y|\}$.

$\mathtt{Ver}(\sigma, Y, m)$**:** Output $1/0 \leftarrow \text{HSSP}.\mathtt{Ver}(pp, W_Y, \sigma)$.

Appendix 1 Digital Signatures

Syntax. A digital signatures scheme consists of the following three polynomial time algorithms. Note that $\mathtt{Ver}$ is deterministic and the others are probabilistic.

Key-Generation $\mathtt{KGen}$**:** It takes a security parameter 1^λ for $\lambda \in \mathbb{N}$, then outputs a public-parameter pp, public-key pk and secret-key sk. $\mathcal{M}$ denotes space of messages. Assume that pp is implicitly inputted to the signing and verification algorithms. $(pp, pk, sk) \leftarrow \mathtt{KGen}(1^\lambda)$

Siging $\mathtt{Sig}$**:** It takes a secret-key sk and a message $M \in \mathcal{M}$, then outputs a signature σ. $\sigma \leftarrow \mathtt{Sig}(sk, M)$

Verification $\mathtt{Ver}$**:** It takes a public-key pk, a message $M \in \mathcal{M}$ and a signature σ, then outputs 1 or 0. $1/0 \leftarrow \mathtt{Ver}(pk, M, \sigma)$

We require every digital signatures scheme to be correct. A scheme is correct if for every $\lambda \in \mathbb{N}$, every $(pp, pk, sk) \leftarrow \mathtt{KGen}(1^\lambda)$, every $M \in \mathcal{M}$, every $\sigma \leftarrow \mathtt{Sig}(sk, M)$, it holds that $1 \leftarrow \mathtt{Ver}(pk, M, \sigma)$.

Security. The most standard unforgeability notion for digital signatures is (weak) existential unforgeability against chosen messages attacks (EUF-CMA) [14]. We consider an experiment for a PPT adversary $\mathcal{A}$ defined as follows.

```
$\boldsymbol{Expt}^{\mathtt{EUF\text{-}CMA}}_{\Sigma_{\mathrm{DS}},\mathcal{A}}(1^\lambda)$:   // $\boldsymbol{Expt}^{\mathtt{EUF\text{-}CMA}}_{\Sigma_{\mathrm{DS}},\mathcal{A}}$
  $(pp, pk, sk) \leftarrow \mathtt{KGen}(1^\lambda)$. $Q := \emptyset$. $(M^* \in \mathcal{M}, \sigma^*) \leftarrow \mathcal{A}^{\mathfrak{Sign}}(pp, pk)$
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  - $\mathfrak{Sign}(M \in \mathcal{M})$:   $\sigma \leftarrow \mathtt{Sig}(sk, M)$. $Q := Q \cup \{M\}$. Rtrn $\sigma$
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  Rtrn 1 if $1 \leftarrow \mathtt{Ver}(pk, M^*, \sigma^*) \wedge M^* \notin Q$.
```

Definition 6. *A digital signatures scheme Σ_{DS} is* `EUF-CMA` *if for any $\lambda \in \mathbb{N}$ and any PPT algorithm $\mathcal{A}$, its advantage $\textit{\textbf{Adv}}^{\textit{EUF-CMA}}_{\Sigma_{\mathrm{DS}},\mathcal{A}}(\lambda) := \Pr[1 \leftarrow \textit{\textbf{Expt}}^{\textit{EUF-CMA}}_{\Sigma_{\mathrm{DS}},\mathcal{A}}(1^{\lambda})]$ is negligible.*

We define the other two unforgeability notions, namely unforgeability against random messages attacks (`UF-RMA`) [13] and unforgeability against extended random messages attacks (`UF-XRMA`) [2]. For `UF-RMA` or `UF-XRMA`, we consider an experiment which is the same as the one for `EUF-CMA` except for the signing oracle. In the experiment for `UF-RMA`, the signing oracle takes no input and returns a signature σ on a randomly-chosen message $M(\xleftarrow{\mathrm{U}} \mathcal{M})$. In the experiment for `UF-XRMA`, the signing oracle returns not only a signature σ on randomly-chosen message $M(\xleftarrow{\mathrm{U}} \mathcal{M})$ but also some information *aux* about the random coins used to select M. The two notions are defined analogously to `EUF-CMA`, cf. Definition 6.

Appendix 2 Non-interactive Witness-Indistinguishable (NIWI) Proof

Syntax. An NIWI system for the NP relation $R : \{0,1\}^* \times \{0,1\}^* \rightarrow 1/0$ consists of the following 3 polynomial-time algorithms. Note that `Ver` is deterministic and the others are probabilistic.

Setup `Setup`**:** It takes a security parameter 1^{λ} for $\lambda \in \mathbb{N}$, then outputs a common reference string (CRS) *crs*. $crs \leftarrow \mathtt{Setup}(1^{\lambda})$

Proving `Pro`**:** It takes the CRS *crs*, a statement $x \in \{0,1\}^*$ and a witness $w \in \{0,1\}^*$, then outputs a proof π. $\pi \leftarrow \mathtt{Pro}(crs, x, w)$

Verification `Ver`**:** It takes the CRS *crs*, a statement $x \in \{0,1\}^*$ and a proof π, then outputs a verification result, which is 1 (accept) or 0 (reject). $1/0 \leftarrow \mathtt{Ver}(crs, x, \pi)$

We require every NIWI system to be correct. An NIWI system is correct if for every $\lambda \in \mathbb{N}$, every $crs \leftarrow \mathtt{Setup}(1^{\lambda})$, every $x \in \{0,1\}^*$, every $w \in \{0,1\}^*$ s.t. $1 \leftarrow R(x, w)$, and every $\pi \leftarrow \mathtt{Pro}(crs, x, w)$, it holds that $1 \leftarrow \mathtt{Ver}(crs, x, \pi)$.

Security. We define two security requirements, namely perfect witness-indistinguishability (`WI`) and perfect witness-extractability (`WE`).

Definition 7. *An NIWI system is perfectly witness-indistinguishable (*`WI`*), if for every $\lambda \in \mathbb{N}$, every $crs \leftarrow \mathtt{Setup}(1^{\lambda})$, every $x \in \{0,1\}^*$, and every $w_0, w_1 \in \{0,1\}^*$ s.t. $1 \leftarrow R(x, w_b)$ for each $b \in \{0,1\}$, $\mathtt{Pro}(crs, x, w_0)$ distributes identically to $\mathtt{Pro}(crs, x, w_1)$.*

Definition 8. *An NIWI system is perfectly witness-extractable (*`WE`*), if for every $\lambda \in \mathbb{N}$, there exist two algorithms* `SimSetup` *and* `Extract` *that satisfy both of the following two conditions.*

1. *For every PPT algorithm $\mathcal{A}$, $\textit{\textbf{Adv}}^{\textit{WE}}_{\Sigma_{\mathrm{NIWI}},\mathcal{A}}(\lambda) := |\Pr[1 \leftarrow \mathcal{A}(crs) \mid crs \leftarrow \mathtt{Setup}(1^{\lambda})] - \Pr[1 \leftarrow \mathcal{A}(crs) \mid (crs, ek) \leftarrow \mathtt{SimSetup}(1^{\lambda})]|$ is negligible.*

2. For every PPT algorithm $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}(crs, ek) \leftarrow \texttt{SimSetup}(1^\lambda); (x, \pi) \leftarrow \mathcal{A}(crs);\\ w \leftarrow \texttt{Extract}(crs, ek, x, \pi) : 1 \leftarrow \texttt{Ver}(crs, x, \pi) \wedge 0 \leftarrow R(x, w)\end{array}\right] = 0.$$

Appendix 3 Waters Signatures [23]

Their scheme is based on a symmetric bilinear paring $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with prime order p and generator $g \in \mathbb{G}$.

$\texttt{KGen}(1^\lambda, L)$**:]** $L \in \mathbb{N}$ denotes bit length of a message. Generate $pp := (h, u', u_0, \cdots, u_{L-1})$, where $h, u', u_0, \cdots, u_{L-1} \xleftarrow{\text{U}} \mathbb{G}$. For each message $m \in \{0,1\}^L$ being parsed as $m[L-1] \parallel \cdots \parallel m[0]$, the programmable hash function $H_\mathbb{G} : \{0,1\}^L \to \mathbb{G}$ takes M then outputs $u' \cdot \prod_{i=0}^{L-1} u_i^{m[i]} \in \mathbb{G}$. Generate $(pk, sk) := (X, x)$, where $X := g^x$ with $x \xleftarrow{\text{U}} \mathbb{Z}_p$. Output (pp, pk, sk).

$\texttt{Sig}(sk, M)$**:** Choose $r \xleftarrow{\text{U}} \mathbb{Z}_p$. Output $\sigma := (h^x \cdot H_\mathbb{G}(m)^r, g^r)$.

$\texttt{Ver}(pk, M, \sigma)$**:** Parse σ as (A, B). Output 1 if $e(A, g) = e(X, h) \cdot e(H_\mathbb{G}(m), B)$. Output 0 otherwise.

Theorem 3. *Waters signatures scheme is* EUF-CMA *under the* CDH *assumption w.r.t. the group $\mathbb{G}$.*

Appendix 4 Abe-Haralambiev-Ohkubo (AHO) SPS [3,4]

Their scheme is based on a symmetric bilinear paring $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with prime order p and generator $g \in \mathbb{G}$.

$\texttt{KGen}(1^\lambda, n)$**:** $n \in \mathbb{N}$ denotes the maximal number of group elements to be signed. Choose generators $g_r, h_r \xleftarrow{\text{U}} \mathbb{G}$. Let $pp := (g_r, g_z)$.
Choose elements $\gamma_z, \delta_z, \alpha_a, \alpha_b \xleftarrow{\text{U}} \mathbb{Z}_p$ and $\gamma_i, \delta_i \xleftarrow{\text{U}} \mathbb{Z}_p$ for $i \in [1, n]$. Compute $g_z := g_r^{\gamma_z}$, $h_z := h_r^{\delta_z}$, $A := e(g_r, g^{\alpha_a})$, $B := e(h_r, g^{\alpha_b})$, and $g_i := g_r^{\delta_i}$ and $h_i := h_r^{\delta_i}$ for $i \in [1, n]$. Output (pp, pk, sk), where $pk := (g_z, h_z, \{g_i, h_i\}_{i=1}^n, A, B)$ and $sk := (\alpha_a, \alpha_b, \gamma_z, \delta_z, \{\gamma_i, \delta_i\}_{i=1}^n)$.

$\texttt{Sig}(sk, (M_1, \cdots, M_n))$**:** Choose $\eta, \rho_a, \rho_b, \omega_a, \omega_b \xleftarrow{\text{U}} \mathbb{Z}_p$. Compute $\theta_1 := g^\eta$ and

$$\theta_2 := g^{\rho_a - \gamma_z \eta} \cdot \prod_{i=1}^n M_i^{\gamma_i}, \quad \theta_3 := g_r^{\omega_a}, \quad \theta_4 := g^{(\alpha_a - \rho_a)/\omega_a},$$
$$\theta_5 := g^{\rho_b - \delta_z \eta} \cdot \prod_{i=1}^n M_i^{\delta_i}, \quad \theta_6 := h_r^{\omega_b}, \quad \theta_7 := g^{(\alpha_b - \rho_b)/\omega_b}.$$

Output a signature $\sigma := (\theta_1, \cdots, \theta_7)$.

$\mathtt{Ver}(pk, (M_1, \cdots, M_n), \sigma)$**:** Output 1 if both of the following two equations hold, namely $A = e(g_z, \theta_1) \cdot e(g_r, \theta_2) \cdot e(\theta_3, \theta_4) \cdot \prod_{i=1}^{n} e(g_i, M_i)$ and $B = e(h_z, \theta_1) \cdot e(h_r, \theta_5) \cdot e(\theta_6, \theta_7) \cdot \prod_{i=1}^{n} e(h_i, M_i)$.

As shown in [3,4], any signature $\sigma = (\theta_1, \cdots, \theta_n)$ can be publicly randomized as follows. The first element is unchanged, i.e., $\theta'_1 := \theta_1$. We choose $\eta_2, \eta_5, \mu, \nu \xleftarrow{\mathrm{U}} \mathbb{Z}_p$ then compute

$$\theta'_2 := \theta_2 \cdot \theta_4^{\eta_2}, \quad \theta'_3 := (\theta_3 \cdot g_r^{-\eta_2})^{1/\mu}, \quad \theta'_4 := \theta_4^{\mu},$$
$$\theta'_5 := \theta_5 \cdot \theta_7^{\eta_5}, \quad \theta'_6 := (\theta_6 \cdot h_r^{-\eta_5})^{1/\nu}, \quad \theta'_7 := \theta_7^{\nu}.$$

According to [3,4], $(\theta'_2, \theta'_3, \theta'_4) \in \mathbb{G}^3$ uniformly distribute under a restriction that $e(g_r, \theta'_2) \cdot e(\theta'_3, \theta'_4) = e(g_r, \theta_2) \cdot e(\theta_3, \theta_4)$, and $(\theta'_5, \theta'_6, \theta'_7) \in \mathbb{G}^3$ uniformly distribute under a restriction that $e(h_r, \theta'_5) \cdot e(\theta'_6, \theta'_7) = e(h_r, \theta_5) \cdot e(\theta_6, \theta_7)$.

Theorem 4. *Let $q \in \mathtt{poly}(\lambda)$ denote the maximal number of signing queries. The signature scheme is EUF-CMA under the q-SFP assumption.*

Appendix 5 Key-Delegatable Predicate Signatures

Key-delegatable predicate signatures (KDPS) consists of the five polynomial-time algorithms. Ver is deterministic and the others are probabilistic.

Setup Setup: It takes 1^{λ}, then outputs a public parameter pp and master-key mk. $\mathbf{X}$, $\mathbf{Y}$ and $\mathcal{M}$ denote the space of key index, signature index and message, respectively. Note that the other algorithms implicitly take pp as input. $(pp, mk) \leftarrow \mathtt{Setup}(1^{\lambda})$

Key-Generation KGen: It takes mk and a key index $X \in \mathbf{X}$, then outputs a secret-key sk. $sk \leftarrow \mathtt{KGen}(mk, X)$

Key-Delegation KDel: It takes a secret-key sk for a key index $X \in \mathbf{X}$ and a key index $X' \in \mathbf{X}$ s.t. $1 \leftarrow \mathcal{P}_{\mathsf{X}}(X, X')$, then outputs a secret-key sk'. $sk' \leftarrow \mathtt{KDel}(sk, X')$

Signing Sig: It takes a secret-key sk for a key index $X \in \mathbf{X}$, a signature index $Y \in \mathbf{Y}$ s.t. $1 \leftarrow \mathcal{P}_{\mathsf{Y}}(X, Y')$ and a message $M \in \mathcal{M}$, then outputs a signature σ. $\sigma \leftarrow \mathtt{Sig}(sk, Y, M)$

Verification Ver: It takes a signature σ, a signature index $Y \in \mathbf{Y}$ and a message $M \in \mathcal{M}$, then outputs 1 or 0. $1/0 \leftarrow \mathtt{Ver}(\sigma, Y, M)$

Every KDPS scheme must be correct. Informally the property means that every correctly generated signature is accepted. Formally the property is defined as follows. A KDPS scheme is correct if $\forall \lambda \in \mathbb{N}$, $\forall (pp, mk) \leftarrow \mathtt{Setup}(1^{\lambda})$, $\forall X \in \mathbf{X}$, $\forall sk \leftarrow \mathtt{KGen}(mk, X)$, $\forall X' \in \mathbf{X}$ s.t. $1 \leftarrow \mathcal{P}_{\mathsf{X}}(X, X')$, $\forall sk' \leftarrow \mathtt{KDel}(sk, X')$, $\forall Y \in \mathbf{Y}$ s.t. $1 \leftarrow \mathcal{P}_{\mathsf{Y}}(X', Y)$, $\forall M \in \mathcal{M}$, $\forall \sigma \leftarrow \mathtt{Sig}(sk', Y, M)$, $1 \leftarrow \mathtt{Ver}(\sigma, Y, M)$ holds.

As security for KDPS, we require unforgeability and signer-privacy. As a notion of unforgeability, we define *(weak) existential unforgeability against adaptively-chosen messages and predicate attack* (EUF-CMA). For a PPT algorithm $\mathcal{A}$, we consider the following experiment.

```
$\boldsymbol{Expt}^{\texttt{EUF-CMA}}_{\Sigma_{\mathrm{KDPS}},\mathcal{A}}(1^\lambda)$:
  1. $(pp, mk) \leftarrow \texttt{Setup}(1^\lambda)$. $(\sigma^*, Y^* \in \mathbf{Y}, M^* \in \mathcal{M}) \leftarrow \mathcal{A}^{\mathfrak{Reveal},\mathfrak{Sign}}(pp)$.

    - $\mathfrak{Reveal}(X \in \mathbf{X})$:  $sk \leftarrow \texttt{KGen}(mk, X)$. $Q := Q \cup \{X\}$. $\textbf{Rtrn}$ $sk$.
    - $\mathfrak{Sign}(X \in \mathbf{X}, Y \in \mathbf{Y}, M \in \mathcal{M})$:  $sk \leftarrow \texttt{KGen}(mk, X)$. $\sigma \leftarrow \texttt{Sig}(sk, M, Y)$.
      $Q' := Q' \cup \{(Y, M, \sigma)\}$. $\textbf{Rtrn}$ $\sigma$.

  2. $\textbf{Rtrn}$ 1 if (1) $1 \leftarrow \texttt{Ver}(\sigma^*, Y^*, M^*)$, (2) $\forall X \in Q$, $0 \leftarrow \mathcal{P}_{\mathsf{Y}}(X, Y^*)$
       and (3) $(Y^*, M^*, \cdot) \notin Q'$.
  3. $\textbf{Rtrn}$ 0.
```

Definition 9. *A KDPS scheme Σ_{KDPS} is* EUF-CMA *if for every $\lambda \in \mathbb{N}$ and every PPT $\mathcal{A}$, $\mathcal{A}$'s advantage $\boldsymbol{Adv}^{\textit{EUF-CMA}}_{\Sigma_{\mathrm{KDPS}},\mathcal{A}}(\lambda) := \Pr[1 \leftarrow \boldsymbol{Expt}^{\textit{EUF-CMA}}_{\Sigma_{\mathrm{KDPS}},\mathcal{A}}(1^\lambda)]$ is negligible.*

As a notion of signer-privacy, we define perfect signer-privacy (PRV). For a probabilistic algorithm $\mathcal{A}$, we consider the following two experiments.

```
$\boldsymbol{Expt}^{\texttt{PRV}}_{\Sigma_{\mathrm{KDPS}},\mathcal{A},0}(1^\lambda)$:  // $\boldsymbol{Expt}^{\texttt{PRV}}_{\Sigma_{\mathrm{KDPS}},\mathcal{A},1}$
  $(pp, mk) \leftarrow \texttt{Setup}(1^\lambda)$. $(pp, mk, \mu) \leftarrow \texttt{SimSetup}(1^\lambda)$.
  $\textbf{Rtrn}$ $b' \leftarrow \mathcal{A}^{\mathfrak{Reveal},\mathfrak{Sign}}(pp, mk)$.

    - $\mathfrak{Reveal}(X \in \mathbf{X})$:
      $sk \leftarrow \texttt{KGen}(mk, X)$. $sk \leftarrow \texttt{SimKGen}(mk, \mu, X)$. $Q := Q \cup \{(X, sk)\}$. $\textbf{Rtrn}$ $sk$.
    - $\mathfrak{Sign}(X \in \mathbf{X}, sk, Y \in \mathbf{Y}, M \in \mathcal{M})$:
      $\textbf{Rtrn}$ $\perp$ if $(X, sk) \notin Q \vee 0 \leftarrow \mathcal{P}_{\mathsf{Y}}(X, Y)$.
      $\sigma \leftarrow \texttt{Sig}(sk, Y, M)$. $\sigma \leftarrow \texttt{SimSig}(mk, \mu, Y, M)$. $\textbf{Rtrn}$ $\sigma$.
```

The latter is associated with 3 polynomial-time algorithms $\{\texttt{SimSetup}, \texttt{SimKGen}, \texttt{SimSig}\}$. The grey parts are considered in the latter, but ignored in the former.

Definition 10. *A KDPS scheme Σ_{KDPS} is perfectly signer-private* (PRV) *if for every $\lambda \in \mathbb{N}$ and every probabilistic algorithm $\mathcal{A}$, there exist polynomial-time algorithms $\{\texttt{SimSetup}, \texttt{SimKGen}, \texttt{SimSig}\}$ such that $\mathcal{A}$'s advantage $\boldsymbol{Adv}^{\textit{PRV}}_{\Sigma_{\mathrm{KDPS}},\mathcal{A}}(\lambda) := |\sum_{b=0}^{1}(-1)^b \Pr[1 \leftarrow \boldsymbol{Expt}^{\textit{PRV}}_{\Sigma_{\mathrm{KDPS}},\mathcal{A},b}(1^\lambda)]|$ is 0.*

Appendix 6 An Omitted Part in the Proof of Lemma 3

We prove that there exists a PPT adversary $\mathcal{B}'_3$ s.t. $\Pr[S_2 \wedge \mathsf{Abort}] \leq 4q \cdot d_M \cdot (L+1) \cdot \mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}'_3,\mathbb{G}}(\lambda)$, where $d_M \in \mathtt{poly}(\lambda)$ denotes the maximal cardinality of the set M.

Let $\mathcal{A}$ denote a PPT adversary which makes the event $S_2 \wedge \mathsf{Abort}$ occurs with a non-negligible probability. Let $\mathcal{B}'_3$ denote a PPT adversary which, by using $\mathcal{A}$ as black-box, attempts to solve the CDH problem relative to the group $\mathbb{G}$. $\mathcal{B}'_3$ behaves as follows.

Receive (g, g^a, g^b) as an instance of the CDH problem. Honestly generate a key-pair $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$ of the AHO scheme and a GS CRS $\boldsymbol{f} = (\vec{f}_1, \vec{f}_2, \vec{f}_3)$ in the perfect soundness setting. Then conduct the following two steps.

1. Set $l := 2d_M$. Choose uniformly at random an integer k satisfying $0 \leq k \leq L$. Assume that $l(L+1) \leq p$.
2. Let $h := g^a$. Choose $x', x_0, \cdots, x_{L-1} \xleftarrow{U} \mathbb{Z}_l$ and $y', y_0, \cdots, y_{L-1} \xleftarrow{U} \mathbb{Z}_p$. For an element $m \in \{0,1\}^L$, define two functions $J, K : \{0,1\}^L \to \mathbb{Z}_p$ as $J(m) := x' + \sum_{i=0}^{L-1} x_i \cdot m[i] - lk$ and $K(m) := y' + \sum_{i=0}^{L-1} y_i \cdot m[i]$. Set $u' := (g^b)^{-lk+x'} \cdot g^{y'}$ and $u_i := (g^b)^{x_i} \cdot g^{y_i}$ for $i \in [0, L-1]$. It holds that $u' \prod_{i=0}^{L-1} u_i^{m[i]} = (g^b)^{-lk+x'+\sum_{i=0}^{L-1} x_i \cdot m[i]} \cdot g^{y'+\sum_{i=0}^{L-1} y_i \cdot m[i]} = (g^b)^{J(m)} \cdot g^{K(m)}$.

Set $pk := (\mathbb{G}, \mathbb{G}_T, g, h, u', \{u_i\}_{i=0}^{L-1}, pk_{\mathsf{s}}, \boldsymbol{f})$ and send it to $\mathcal{A}$. Since we have assumed that the event $S_2 \wedge \mathsf{Abort}$ occurs, it will hold that $\exists \kappa \in [1, q]$ s.t. $(X_\kappa, Y_\kappa) = (X^*, Y^*) \wedge M_\kappa \not\supseteq M^*$. $\mathcal{B}'_3$ guesses such an index κ and chooses $\kappa_{guess} \xleftarrow{U} [1, q]$. The guess is correct at least with probability $1/q$. $\mathcal{B}'_3$ proceeds under an assumption that the guess is correct. When $\mathcal{A}$ issues a query to the signing oracle, $\mathcal{B}'_3$ behaves as follows.

$\mathfrak{Sign}(M, W)$**:** Assume that this query is the κ-th query to the oracle. $\mathcal{B}'_3$ considers the following two cases.

1. $\kappa \neq \kappa_{guess}$**:** $\mathcal{B}'_3$ honestly generates the whole HSSM signature σ oneself, then returns it to $\mathcal{A}$.

2. $\kappa = \kappa_{guess}$**:** If $\exists m \in M$ s.t. $J(m) = 0 \pmod{l}$, $\mathcal{B}'_3$ aborts the simulation. Otherwise, every $m \in M$ satisfies $J(m) \neq 0 \pmod{l}$, which implies $J(m) \neq 0 \pmod{p}$. $\mathcal{B}'_3$ sets $X := g^b$. Then, for each $m \in M$, $\mathcal{B}'_3$ behaves as follows. We have assumed that it holds that $J(m) \neq 0 \pmod{p}$. $\mathcal{B}'_3$ chooses $\chi_m \xleftarrow{U} \mathbb{Z}_p$, then generates $(\sigma_{m,1}, \sigma_{m,2}) := ((g^a)^{\frac{K(m)}{J(m)}}(u' \prod_{i=0}^{L-1} u_i^{m[i]})^{\chi_m}, (g^a)^{-\frac{1}{J(m)}} g^{\chi_m})$. Let $\chi'_m := \chi_m - \frac{a}{J(m)}$. Obviously, $\sigma_{m,2} = g^{\chi'_m}$. It holds that $\sigma_{m,1} = g^{ab} \cdot \{(g^b)^{J(m)} \cdot g^{K(m)}\}^{-\frac{a}{J(m)}} \cdot \{(g^b)^{J(m)} \cdot g^{K(m)}\}^{\chi_m} = g^{ab} \cdot \{(g^b)^{J(m)} \cdot g^{K(m)}\}^{\chi'_m} = h^b \cdot H_{\mathbb{G}}(m)^{\chi'_m}$. Thus, the Waters signature $(\sigma_{m,1}, \sigma_{m,2})$ correctly distributes. $\mathcal{B}'_3$ honestly generates the other elements of the HSSM signature σ, then returns σ to $\mathcal{A}$.

$\mathcal{B}'_3$ receives a forged signature σ^* sent by $\mathcal{A}$. Since we have assumed that the event $S_2 \wedge \mathsf{Abort}$ occurs, all of the following three conditions hold, namely (a) $X^* = X_{\kappa_{guess}}$, (b) $M^* \not\subseteq M_{\kappa_{guess}}$, and (c) *for each* $m \in M^*$, $(\sigma^*_{m,1}, \sigma^*_{m,2}) = (h^b \cdot H_{\mathbb{G}}(m)^{\chi_m}, g^{\chi_m})$ *with some* $\chi_m \in \mathbb{Z}_p$.

The second condition (b) implies that $\exists m^* \in M^*$ s.t. $m^* \notin M_{\kappa_{guess}}$. $\mathcal{B}'_3$ arbitrarily chooses a single element m^* satisfying the above condition, then aborts the simulation if $J(m^*) \neq 0 \pmod{p}$. $\mathcal{B}'_3$ computes $\sigma^*_{m^*,1}/(\sigma^*_{m^*,2})^{K(m^*)} = h^b \cdot H_{\mathbb{G}}(m^*)^{\chi_{m^*}}/(g^{K(m^*)})^{\chi_{m^*}} = h^b = g^{ab} \in \mathbb{G}$ then outputs it as the answer to the CDH problem.

Let $\mathsf{SimAbort}$ denote the event that $\mathcal{B}'_3$ aborts the simulation. At least when the event $S_2 \wedge \mathsf{Abort}$ has occurred and $\mathcal{B}'_3$ has not aborted the simulation, $\mathcal{B}'_3$ finds the correct answer to the CDH problem. Thus, it holds that

$$\begin{aligned}\mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}'_3,\mathbb{G}}(\lambda) &\geq \Pr[S_2 \wedge \mathsf{Abort} \wedge \neg\mathsf{SimAbort}] \\ &= \Pr[\neg\mathsf{SimAbort} \mid S_2 \wedge \mathsf{Abort}] \cdot \Pr[S_2 \wedge \mathsf{Abort}],\end{aligned}$$

which implies that

$$\Pr[S_2 \wedge \mathsf{Abort}] \leq \frac{1}{\Pr[\neg\mathsf{SimAbort} \mid S_2 \wedge \mathsf{Abort}]} \cdot \mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}'_3,\mathbb{G}}(\lambda).$$

We analyze the probability $\Pr[\neg\mathsf{SimAbort} \mid S_2 \wedge \mathsf{Abort}]$. We define three events.

E_1**:** $(X^*, Y^*) = (X_{\kappa_{guess}}, Y_{\kappa_{guess}})$
E_2**:** $\forall m \in M_{\kappa_{guess}}$, $J(m) \neq 0 \pmod{l}$
E_3**:** $\exists m^* \in M^*$ s.t. $m^* \notin M_{\kappa_{guess}} \wedge J(m^*) = 0 \pmod{p}$

We obtain

$$\begin{aligned}&\Pr[\neg\mathsf{SimAbort} \mid S_2 \wedge \mathsf{Abort}] \\ &= \Pr[\mathsf{E}_1 \wedge \mathsf{E}_2 \wedge \mathsf{E}_3 \mid S_2 \wedge \mathsf{Abort}] \\ &= \Pr[\mathsf{E}_1 \mid S_2 \wedge \mathsf{Abort}] \cdot \Pr[\mathsf{E}_2 \wedge \mathsf{E}_3 \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1] \\ &= \Pr[\mathsf{E}_1 \mid S_2 \wedge \mathsf{Abort}] \cdot \Pr[\mathsf{E}_3 \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1] \cdot \Pr[\mathsf{E}_2 \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1 \wedge \mathsf{E}_3].\end{aligned}$$

We analyze each term. Obviously, $\Pr[\mathsf{E}_1 \mid S_2 \wedge \mathsf{Abort}] \geq 1/q$. The second term is analyzed as follows.

$$\begin{aligned}&\Pr[\mathsf{E}_3 \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1] \\ &= \Pr[J(m^*) = 0 \pmod{p} \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1] \\ &= \Pr[J(m^*) = 0 \pmod{l} \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1] \\ &\quad \cdot \Pr[J(m^*) = 0 \pmod{p} \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1 \wedge J(m^*) = 0 \pmod{l}] \\ &= \frac{1}{l}\frac{1}{L+1}\end{aligned}$$

The third term is as follows.

$$\begin{aligned}&\Pr[\mathsf{E}_2 \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1 \wedge \mathsf{E}_3] \\ &= \Pr\left[\bigwedge_{m \in M_{\kappa_{guess}}} J(m) \neq 0 \pmod{l} \middle| S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1 \wedge \mathsf{E}_3\right] \\ &\geq 1 - \sum_{m \in M_{\kappa_{guess}}} \Pr[J(m) = 0 \pmod{l} \mid S_2 \wedge \mathsf{Abort} \wedge \mathsf{E}_1 \wedge \mathsf{E}_3] \\ &= 1 - \sum_{m \in M_{\kappa_{guess}}} \frac{1}{l} \geq 1 - \frac{d_M}{l}\end{aligned}$$

As a result, we obtain

$$\Pr[\neg\mathsf{SimAbort} \mid S_2 \wedge \mathsf{Abort}] \geq \frac{1}{q} \cdot \frac{1}{l} \cdot \frac{1}{L+1} \cdot \left(1 - \frac{d_M}{l}\right) = \frac{1}{4q \cdot d_M \cdot (L+1)}$$

Therefore, $\Pr[S_2 \wedge \mathsf{Abort}] \leq 4q \cdot d_M \cdot (L+1) \cdot \mathtt{Adv}^{\mathtt{CDH}}_{\mathcal{B}'_3,\mathbb{G}}(\lambda)$. □

References

1. Abdalla, M., Catalano, D., Dent, A.W., Malone-Lee, J., Neven, G., Smart, N.P.: Identity-based encryption gone wild. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 300–311. Springer, Heidelberg (2006). https://doi.org/10.1007/11787006_26
2. Abe, M., Chase, M., David, B., Kohlweiss, M., Nishimaki, R., Ohkubo, M.: Constant-size structure-preserving signatures: generic constructions and simple assumptions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 4–24. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_3
3. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_12
4. Abe, M., Haralambiev, K., Ohkubo, M.: Signing on elements in bilinear groups for modular protocol design. Cryptology ePrint Archive (2010)
5. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 1–20. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_1
6. Attrapadung, N., Libert, B., Peters, T.: Computing on authenticated data: new privacy definitions and constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 367–385. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_23
7. Attrapadung, N., Libert, B., Peters, T.: Efficient completely context-hiding quotable and linearly homomorphic signatures. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 386–404. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_24
8. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE SP 2007, pp. 321–334. IEEE (2007)
9. Blömer, J., Eidens, F., Juhnke, J.: Enhanced security of attribute-based signatures. In: Camenisch, J., Papadimitratos, P. (eds.) CANS 2018. LNCS, vol. 11124, pp. 235–255. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00434-7_12
10. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_3
11. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
12. Bultel, X., Lafourcade, P., Lai, R.W.F., Malavolta, G., Schröder, D., Thyagarajan, S.A.K.: Efficient invisible and Unlinkable Sanitizable signatures. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11442, pp. 159–189. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17253-4_6
13. Even, S., Goldreich, O., Micali, S.: On-line/off-line digital signatures. J. Cryptol. **9**(1), 35–67 (1996). https://doi.org/10.1007/BF02254791
14. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)
15. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_24

16. Katz, J., Maffei, M., Malavolta, G., Schröder, D.: Subset predicate encryption and its applications. In: Capkun, S., Chow, S.S.M. (eds.) CANS 2017. LNCS, vol. 11261, pp. 115–134. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02641-7_6
17. Kiltz, E., Mityagin, A., Panjwani, S., Raghavan, B.: Append-only signatures. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 434–445. Springer, Heidelberg (2005). https://doi.org/10.1007/11523468_36
18. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_4
19. Lewko, A., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_27
20. Libert, B., Joye, M., Yung, M., Peters, T.: Secure efficient history-hiding append-only signatures in the standard model. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 450–473. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_20
21. Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 376–392. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19074-2_24
22. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: Kim, K. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 285–304. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45861-1_22
23. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_7
24. Zhang, F., Kim, K.: ID-based blind signature and ring signature from pairings. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 533–547. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36178-2_33

Adaptively Secure Identity-Based Encryption from Middle-Product Learning with Errors

Jingjing Fan[1], Xingye Lu[2](✉), and Man Ho Au[2]

[1] University of Hong Kong, Pok Fu Lam, China
jjfan@cs.hku.hk

[2] The Hong Kong Polytechnic University, Hung Hom, China
{xing-ye.lu,mhaau}@polyu.edu.hk

Abstract. Introduced in 2017, Middle-Product Learning with Errors (MPLWE) and its variants offer a way to construct cryptographic primitives which preserve the efficiency of those based on Polynomial-LWE (PLWE) while being at least as secure as them over a broad choice of number fields. Based on MPLWE, a series of cryptographic schemes have been proposed, including public key encryption (PKE), digital signature, and identity-based encryption (IBE). In this paper, we extend this line of research and propose a new IBE scheme that is adaptively secure in the standard model from MPLWE. Existing IBE schemes from MPLWE only offer selective security or rely on the random oracle model. We follow the blueprint of Agrawal et al. at EUROCRYPT2010 and adapt the well-known partitioning technique to the MPLWE setting. The resulting scheme offers similar efficiency to schemes based on PLWE under a milder assumption.

Keywords: Lattice-Based Cryptography · Middle-Product LWE · Identity-Based Encryption

1 Introduction

Conceptualised by Shamir in 1984 [24], identity-based encryption (IBE) allows users to use any string (e.g. email address) as the public key. However, it was not until two decades later that the first realization of IBE was proposed. In 2001, Boneh and Franklin [7] and Cocks [10] proposed IBE schemes from Weil pairing and quadratic residues for a composite modulus respectively. Both schemes are proven secure in the random oracle model. Subsequently, [5,8] presented IBE in the standard model offering selective security, meaning that the adversaries are required to declare the challenge identity before seeing the public parameters. More recently, adaptively secure IBE schemes in the standard model have been developed [6,26,27] based on the hardness of certain number-theoretic problems.

The advent of quantum computers poses a serious threat to the aforementioned schemes as quantum algorithms can efficiently solve the number-theoretic

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 320–340, 2023.
https://doi.org/10.1007/978-3-031-35486-1_15

problems on which they rely. To prevent attacks from quantum computers, post-quantum cryptography is being actively investigated. Lattice-based cryptography is one of the most important area of post-quantum cryptography. Based on the hardness of the Short Integer Solution problem (SIS) introduced in [2] and the Learning With Errors problem (LWE) introduced in [21], lattice-based cryptography has led to numerous cryptographic constructions that are conjectured to be hard to break even for quantum algorithms. In 2008, Gentry et al. proposed the first lattice-based IBE scheme [12] in the random oracle model. In 2010, Agrawal et al. [1] proposed an IBE scheme that is adaptively secure in the standard model. Cash et al. [9] introduced a new lattice-based admissible hash called a Bonsai tree and used it to construct a hierarchical identity-based encryption scheme in the standard model. These IBE schemes [1,9] are both from standard lattices and require $O(\ell)$ basic matrices in the master public keys, where ℓ is the length of the identity. Later, lattice-based IBE schemes with compact master public keys were proposed. In 2016, Yamada [28] presented adaptively secure IBE schemes with short master public keys from standard LWE. In the same year, Katsumata and Yamada [14], proposed adaptively secure compact IBE schemes from ring LWE. Both of these two schemes have master public keys with around $O(\sqrt{\ell})$ basic matrices (vectors). To further reduce the master public key size, Zhang et al. [30] and Yamada [29] proposed adaptively secure IBE from standard LWE with master public key size grows logarithmically with the identity length. Among these compact IBE schemes, the ones from standard lattice have large master public key sizes, while those from ideal lattice are based on RLWE problems whose hardness is relatively less well-understood. The middle-product learning with errors (MPLWE) was proposed by Roşca et al. [22]. It is a variant of the polynomial learning with errors (PLWE) [25] and the ring learning with errors (RLWE) [18] problem. It exploits the middle-product of polynomials modulo prime q. MPLWE is shown to be as secure as PLWE for a large class of polynomials. Roşca et al. also presented an encryption scheme from MPLWE, whose efficiency is similar to those based on RLWE. MPLWE has also been used to construct other cryptographic primitives such as digital signatures [3] and ring signatures [11]. In 2019, Lombardi et al. generalized MPLWE to degree-parameterized MPLWE [16], and showed that schemes based on degree-parameterized MPLWE are at least as secure as those based on PLWE. They gave a pre-image sampleable function for MPLWE (can be regarded as a GPV [12] sampler in the MPLWE setting) and used it to construct IBE schemes in the random oracle and standard model respectively. In 2020, Le et al. [15] improved the IBE construction and proposed a hierarchical IBE scheme. While these schemes [15,16] are secure in the standard model, they only offer selective security. A natural problem is, therefore, to construct adaptively secure IBE in the standard model from MPLWE.

1.1 Our Contribution

In this paper, we focus on the construction of adaptively secure IBE based on degree-parameterized middle-product LWE (dMPLWE) in the standard model.

We follow the blueprint of [1] (which makes use of abort-resistant hash functions) and adapt it to the MPLWE setting. This enables us to go from the selectively secure IBE to an adaptively secure one. Compared with existing adaptively secure IBE from ideal lattices [14,28], our dMPLWE IBE benefits from the fact that its security does not rely on the choice of the underlying ring, while enjoying similar efficiency. We compared our IBE scheme with existing lattice-based adaptively secure IBE schemes in the standard model in Table 1.

Remarks. We set identity length $\ell = O(n)$. KY16 type 2 [14] has the smallest master public key and ciphertext/secret key size among the existing adaptively secure IBEs from RLWE, while the Yam17 type 2 [29] has the smallest master public key size among the ones from standard LWE. Our scheme is the first and only scheme from MPLWE. At similar efficiency, our scheme would be more secure compared with those from RLWE. Compared with schemes from standard lattice, our scheme typically enjoys shorter master public key size. Also, our IBE scheme is secure when the LWE assumption holds for $1/\alpha = \tilde{O}(n^3)$ while Yam17 Type 2 [29] is secure only when LWE assumption holds with $1/\alpha = \mathsf{poly}(n)$ for all polynomials $\mathsf{poly}(n)$.

Table 1. Comparison between Lattice-Base Adaptively Secure IBEs in the Standard Model and Our Scheme

Scheme	$\lvert\mathsf{mpk}\rvert$	$\lvert C\rvert, \lvert\mathsf{sk}_{\mathsf{id}}\rvert$	$1/\alpha$ for LWE Assumption	LWE type
CHKP10 [9]	$O(n^2\ell\log n)$	$O(n\ell\log n)$	$\tilde{O}(n^{1.5})$	Standard LWE
ABB10 [1]	$O(n^2\ell\log n)$	$O(n\log n)$	$\tilde{O}(n^2)$	Standard LWE
Yam16: Type 1 [28]	$O(n^2\ell^{\frac{1}{c}}\log n)$	$O(n\log n)$	$\mathsf{superpoly}(n)$	Standard LWE
Yam16: Type 2 [28]	$O(n^2\ell^{\frac{1}{c}}\log n)$	$O(n\log n)$	All $\mathsf{poly}(n)$	Standard LWE
KY16: Type 1 [14]	$O(n\ell^{\frac{1}{c}}\log n)$	$O(n\log n)$	$\tilde{O}(n^{2c-\frac{1}{2}})$	RLWE
KY16: Type 2 [14]	$O(n\ell^{\frac{1}{c}})$	$O(n)$	$\tilde{O}(n^{2c+\frac{3}{8}})$	RLWE
ZCZ16 [30]	$O(n^2\log\ell\log n)$	$O(n\log n)$	$Q^2\tilde{O}(n^{6.5})$	Standard LWE
Yam17: Type 1 [29]	$O(n^2\log^3\ell\log n)$	$O(n\log n)$	$\tilde{O}(n^{11})$	Standard LWE
Yam17: Type 2 [29]	$O(n^2\log^2\ell\log n)$	$O(n\log n)$	All $\mathsf{poly}(n)$	Standard LWE
Our IBE Scheme	$O(n\ell\log n)$	$O(n\log n)$	$\tilde{O}(n^3)$	MPLWE

"$1/\alpha$" for LWE assumption refers to the underlying LWE assumption used in the security reduction. ℓ is the identity length. c is a flexible constant (recommended to be 2 or 3 in [14]). "All $\mathsf{poly}(n)$" means that we have to assume the LWE assumption for all polynomial. $Q = \mathsf{poly}(n)$ refers to the number of queries made by the adversary. $\tilde{O}(f(n)) = O(f(n)\log^c n)$ for some constant c.

1.2 Overview

To illustrate our construction, we first provide a high-level overview. Similar to the adaptively secure IBE schemes based on LWE/PLWE, our approach employs a hash function, denoted as $h(\cdot)$, to map each user identity to a vector with polynomial entries. This function enables us to associate each identity id with a public polynomial vector $[\overrightarrow{a}|h(\mathsf{id})] \in (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau} \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, where $\overrightarrow{a}$ and the polynomial vectors used to compute function h served as master public keys. The polynomial vector $\overrightarrow{a}$ is equipped with a trapdoor that facilitates the sampling of $\overrightarrow{t}$ such that $\overrightarrow{a} \cdot \overrightarrow{t} = w$ for a given w. To extract the secret key for a user id, the KGC utilizes the trapdoor of a to generate $\mathsf{sk}_{\mathsf{id}}$ such that $[\overrightarrow{a}|h(\mathsf{id})] \cdot \mathsf{sk}_{\mathsf{id}} = u$ for a predetermined u in the master public key. To encrypt a message, one utilizes $[\overrightarrow{a}|h(\mathsf{id})]$ and u as the public key and follows the Dual Regev PKE [12] in the MPLWE setting.

During the proof, $\overrightarrow{a}$ is randomly sampled from its distribution, without a known trapdoor, while $h(\cdot)$ is generated in a way such that for a large set of identities, the output vector polynomial is equipped with a trapdoor that enables the sampling of user secret keys. For the remaining identities, there exists no such trapdoor. We then proceed to demonstrate that the probability that all the key extraction query identities belong to the large set, while the challenge identity belongs to the remaining set, is non-negligible (the argument makes use of the aforementioned abort-resistant hash functions). Finally, we leverage the adversary's response in this setting to distinguish between MPLWE samples and those uniformly distributed ones.

Paper Organization. We give preliminaries in Sect. 2. We present the trapdoor generation and sampling algorithms in Sect. 3, followed by our construction and security analysis in Sect. 4. Finally, we conclude our work in Sect. 5.

2 Preliminaries

2.1 Notations

We use n to denote the security parameter. We use standard big-O notation to classify the growth of functions. We say that $f(n) = \tilde{O}(g(n))$ if $f(n) = O(g(n) \cdot \log^c n)$ for some constant c. We use $\mathsf{poly}(n)$ to denote a random function $f(n) = O(n^c)$ for some constant c. We say that a function $f(n)$ is negligible (denoted by $\mathsf{negl}(n)$) if for every polynomial p, there exists an N s.t. for all $n > N$, it holds that $f(n) < \frac{1}{p(n)}$. We say that a probability is overwhelming if it is $1 - \mathsf{negl}(n)$.

2.2 Identity-Based Encryption

Syntax. An identity-based encryption(IBE) scheme usually consists of four algorithms, namely $(\mathsf{KeyGen}, \mathsf{Extract}, \mathsf{Enc}, \mathsf{Dec})$, which run as follows:

- $\mathsf{KeyGen}(1^n) \to (\mathsf{mpk}, \mathsf{msk})$: On input the security parameter 1^n, generates master public key and master secret key pair $(\mathsf{mpk}, \mathsf{msk})$.
- $\mathsf{Extract}(\mathsf{mpk}, \mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$: On input the master public key mpk, master secret key msk and the identity id, outputs the secret key for the identity $\mathsf{sk}_{\mathsf{id}}$.

- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m) \rightarrow c$: On input the master public key mpk, identity id and message μ, outputs the ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c) \rightarrow m$: On input the secret key for the identity $\mathsf{sk}_{\mathsf{id}}$ and the ciphertext c, outputs a message m.

An IBE scheme $\mathsf{IBE} = (\mathsf{KeyGen}, \mathsf{Extract}, \mathsf{Enc}, \mathsf{Dec})$ possesses correctness if for all identities id and message m, we have $\Pr[\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)) = m] = 1 - \mathsf{negl}(n)$ where the probability is taken with respect to the randomness of KeyGen, $\mathsf{Extract}$, Enc, and Dec.

IND-CPA Security. The adaptive IND-CPA security for an IBE scheme can be defined by the following game between adversary $\mathcal{A}$ and challenger $\mathcal{C}$.

- **Setup.** Challenger $\mathcal{C}$ runs $\mathsf{KeyGen}(1^n) \rightarrow (\mathsf{mpk}, \mathsf{msk})$ and sends mpk to adversary $\mathcal{A}$.
- **Key Extraction Query** Adversary $\mathcal{A}$ sends a user ID id to challenger $\mathcal{C}$, $\mathcal{C}$ returns $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{Extract}(\mathsf{mpk}, \mathsf{msk}, \mathsf{id})$.
- **Challenge.** Adversary $\mathcal{A}$ picks the challenging identity id^* and two messages m_0 and m_1 on which it wishes to be challenged. If id^* has been queried in key extraction query, $\mathcal{C}$ aborts and outputs $\perp$. Otherwise, $\mathcal{C}$ tosses a coin $\mathsf{coin} \overset{\$}{\leftarrow} \{0, 1\}$ and sends $C^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}^*, m_{\mathsf{coin}})$ to $\mathcal{A}$.
- **Key Extraction Query.** Adversary $\mathcal{A}$ can make further key extraction query on identity $\mathsf{id} \neq \mathsf{id}^*$.
- **Output.** $\mathcal{A}$ outputs a coin coin^*.

$\mathcal{A}$ wins the game if $\mathsf{coin}^* = \mathsf{coin}$. The advantage of $\mathcal{A}$ in this IND-CPA game is defined by $Adv_{\mathcal{A}} = \Pr[\mathcal{A} \text{ wins IND-CPA game}] - \frac{1}{2}$.

Definition 1 (IND-CPA). *An IBE scheme is adaptive IND-CPA secure if for any polynomial-time adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{\mathsf{IBE}}$ is negligible.*

2.3 Lattices and Gaussian Distributions

An n-dimensional lattice in m-dimensional Euclidean space is an additive discrete set defined as follows: $\Lambda(\mathbf{b_1}, \ldots, \mathbf{b_n}) = \{\sum_{i=1}^{n} x_i \mathbf{b_i} | x_i \in \mathbb{Z}\}$, where $\mathbf{b_1}, \ldots, \mathbf{b_n} \in \mathbb{Z}^m$ are linearly independent column vectors. We say that Λ is full rank iff $n = m$. The dual lattice of Λ, denoted as Λ^* is defined as: $\Lambda^* = \{\mathbf{y} \in \text{span}_{\mathbb{R}}(\Lambda) | \langle \mathbf{y}, \mathbf{x} \rangle \in \mathbb{Z}, \forall \mathbf{x} \in \Lambda\}$. For positive integers n, q and any matrix $\mathbf{A} \in \mathbb{Z}^{n \times m}$, let $\Lambda^{\perp} := \{\mathbf{z} \in \mathbb{Z}^m | \mathbf{Az} = 0 \mod q\}$. For $\mathbf{u} \in \mathbb{Z}_q^n$ s.t. $\exists \mathbf{t} \in \mathbb{Z}_q^m$ satisfying $\mathbf{At} = \mathbf{u}$, let $\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}) := \{\mathbf{z} \in \mathbb{Z}^m | \mathbf{Az} = \mathbf{u} \mod q\}$, $\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}) = \Lambda^{\perp}(\mathbf{A}) + \mathbf{t}$.

Next, we will recap some well-known definitions concerning Gaussian distribution that would be used in our proof.

Continuous Gaussian Distribution. For a positive semi-definite matrix $\Sigma \in \mathbb{R}^{n \times n}$, the continuous Gaussian distribution D_{Σ} is the probability distribution over $\mathbb{R}^n$ whose density is proportional to $\rho_{\Sigma}(x) = \exp(-\pi x^T \Sigma^{-1} x)$.

Discrete Gaussian Distribution. Given a countable set $S \subset \mathbb{R}^n$ and $s > 0$, the discrete Gaussian distribution $D_{S,\sigma,\mathbf{c}}$ is the probability distribution over S

whose density is proportional to $\rho_{\sigma,\mathbf{c}(x)} := \exp(\frac{-\pi\cdot\|\mathbf{x}-\mathbf{c}\|^2}{\sigma^2})$. That is, for $x \in S$, $D_{S,\sigma,\mathbf{c}} := \frac{\rho_{\sigma,\mathbf{c}}(x)}{\rho_{\sigma,\mathbf{c}}(S)}$. If $\mathbf{c} = 0$, we can omit $\mathbf{c}$ and write $D_{S,\sigma}$ instead.
For any n-dimensional lattice Λ and real $\epsilon > 0$, the smoothing parameter $\eta_\epsilon(\lambda)$ is defined to be the smallest real $s > 0$ such that $\rho_{1/s}(\Lambda^* \setminus \{\mathbf{0}\}) \le \epsilon$ [20].

Gaussian Tail Inequality. For any $\epsilon > 0$, any $\sigma \ge \eta_\epsilon(\mathbb{Z})$, and any $t > 0$, we have $\Pr_{x\leftarrow D_{\mathbb{Z},\sigma,c}}[|x-c| \ge t\cdot\sigma] \le 2e^{-\pi t^2}\cdot\frac{1+\epsilon}{1-\epsilon}$. In particular, for $\epsilon \in (0, 1/2)$ and $t \ge \omega(\sqrt{\log n})$, the probability that $|x-c| \ge t\cdot\sigma$ is negligible in n [12].

2.4 Polynomials, Matrices and Middle Product of Polynomials

In this paper, a vector will refer to a column vector unless otherwise stated. For a matrix $\mathbf{A}$, we will use $A_{i,j}$ to denote its (i,j)-th entry. Let R be a ring. For any $d > 0$ and any set $S \subset R$, we let $S^{<d}[x]$ denote the set of polynomials in $R[x]$ of degree $< d$ whose coefficients are in S. For any distribution χ defined over R, let $\chi^d[x]$ denote the distribution on polynomials in $R^{<d}[x]$ where each coefficient is sampled independently according to χ.

Given a polynomial $a = \sum_{i=0}^{d-1} a_i x^i \in R^{<d}[x]$, define the coefficient vector of a as $\mathbf{a} := (a_0, \cdots, a_{d-1})^T \in R^d$ and set $\overline{\mathbf{a}} = (a_{d-1}, a_{d-2}, \cdots, a_1, a_0)^T \in R^d$. We can see that $\overline{\overline{\mathbf{a}}} = \mathbf{a}$. In particular, for any $0 \le i \le d-1$, a_i denoted the coefficient of x^i in a. Let $\mathbf{d} \in \mathbb{Z}^n = (d_1, \cdots, d_n)^T$ be an integer vector with positive coefficients. Define a column vector, called polynomial vector, $\overrightarrow{a} := (a_1, \cdots, a_n)^T \in R^{<\mathbf{d}}[x]$, where a_i is a polynomial in $R[x]^{<d_i}$ for $i = 1, \cdots, n$.

Let $\mathbf{d} \in \mathbb{Z}^n = (d_1, \cdots, d_n)^T$ be an integer vector with positive coefficients and t be a positive integer. Define polynomial matrix $\overrightarrow{A} := [\overrightarrow{a_1}, \cdots, \overrightarrow{a_t}]^T \in R^{t\times<\mathbf{d}}[x]$, where $\overrightarrow{a_i}$ is a polynomial vector in $R[x]^{<\mathbf{d}}[x]$ for $i = 1, \cdots, t$.

Then, we will define the inner product of polynomial vectors as follows.

Definition 2. *Let* $\mathbf{d}, \mathbf{k} \in \mathbb{Z}^n$ *be integer vectors with positive coefficients. Let* $\overrightarrow{a} := (a_1, \cdots, a_n)^T \in R^{<\mathbf{d}}[x]$, $\overrightarrow{b} := (b_1, \cdots, b_n)^T \in R^{<\mathbf{k}}[x]$ *be polynomial vectors, where* a_i *is a polynomial in* $R^{<d_i}[x]$ *for* $i = 1, \cdots, n$ *and where* b_i *is a polynomial in* $R^{<k_i}[x]$ *for* $i = 1, \cdots, n$. *We define the inner product of* $\overrightarrow{a}$ *and* $\overrightarrow{b}$ *as follows:* $\overrightarrow{a} \cdot \overrightarrow{b} := \sum_{i=1}^n a_i b_i$

We will define the multiplication between polynomial vectors and polynomial matrices as follows.

Definition 3. *Let* $\mathbf{d}, \mathbf{k} \in \mathbb{Z}^n$ *be integer vectors with positive coefficients and* t *be a positive integer. Let* $\overrightarrow{A} := [\overrightarrow{a_1}, \cdots, \overrightarrow{a_t}]^T \in R^{t\times<\mathbf{d}}[x]$ *be a polynomial matrix consisting of* t *polynomial vectors,* $\overrightarrow{a_1}, \cdots, \overrightarrow{a_i}, \cdots \overrightarrow{a_t}$ *in* $R^{<\mathbf{d}}[x]$. *Let* $\overrightarrow{b} := (b_1, \cdots, b_n)^T \in R^{<\mathbf{k}}[x]$ *be a polynomial vector, where* b_i *are polynomials in* $R[x]^{<k}$ *for* $i = 1, \cdots, n$. *We define the matrix-vector multiplication between* $\overrightarrow{A}$ *and* $\overrightarrow{b}$ *as follows:* $\overrightarrow{A} \cdot \overrightarrow{b} = [\overrightarrow{a_1} \cdot \overrightarrow{b}, \cdots, \overrightarrow{a_t} \cdot \overrightarrow{b}]^T$

For a vector $\mathbf{v} \in \mathbb{R}^n$, let $\|\mathbf{v}\|, \|\mathbf{v}\|_\infty$ denote its Euclidean and sup norm respectively. We define the largest singular value of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ as $\sigma_1(\mathbf{A}) := max_{\|\mathbf{u}\|=1}\|\mathbf{Au}\|$.

Lemma 1. *For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, we have $\sigma_1(\mathbf{A}) \leq \sqrt{mn}\max_{i,j}|A_{i,j}|$*

Definition 4 (Tensor Product). *For $m_1, m_2, n_1, n_2 > 0$, the tensor product $\otimes : \mathbb{R}^{m_1\times n_1} \times \mathbb{R}^{m_2\times n_2} \mapsto \mathbb{R}^{m_1m_2\times n_1n_2}$ is defined to be the map:*

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} A_{1,1}\mathbf{B} & \dots & A_{1,n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ A_{m,1}\mathbf{B} & \dots & A_{m,n}\mathbf{B} \end{bmatrix} \text{ for matrix } \mathbf{A} \in \mathbb{R}^{m_1\times n_1} \text{ and matrix } \mathbf{B} \in \mathbb{R}^{m_2\times n_2}.$$

Definition 5 ([17], Definition 6). *Let R be a ring and $d, k > 0$ be positive integers. For any polynomial $a \in R^{<k}[x]$ of degree less than k, let $\mathsf{T}^{k,d}(a)$ denote the matrix in $R^{(k+d-1)\times d}$ whose ith column, for $i = 1, \cdots, d$ is given by the coefficients of $x^{i-1}\cdot a$, listed from the lowest to the highest degree.*

Lemma 2 ([17], Lemma 7). *For $l, k, d > 0$, $a \in R^{<k}[x], b \in R^{<l}[x], \mathsf{T}^{k,l+d-1}(a)\cdot\mathsf{T}^{l,d}(b) = \mathsf{T}^{l+k-1,d}(a\cdot b)$.*

Lemma 3 (Noise Rerandomization [14], Lemma 1). *Let q, l, m be positive integers and r a positive real number satisfying $r > \max\{\omega(\sqrt{\log m}), \omega(\sqrt{\log l})\}$. Let $\mathbf{b} \in \mathbb{Z}_q^m$ be arbitrary and $\mathbf{x}$ chosen from $D_{\mathbb{Z}^m,r}$. Then for any $\mathbf{V} \in \mathbb{Z}^{m\times l}$ and positive real $\sigma > \sigma_1(\mathbf{V})$, there exists a PPT algorithm $\mathsf{ReRand}(\mathbf{V}, \mathbf{b}+\mathbf{x}, r, \sigma)$ that outputs $\mathbf{b}' = \mathbf{b}\mathbf{V} + \mathbf{x}' \in \mathbb{Z}_q^l$ where $\mathbf{x}'$ is distributed statistically close to $D_{\mathbb{Z}^l,2r\sigma}$.*

For our purpose, we focus on the polynomials with $\mathsf{poly}(n)$-bounded expansion factors. One such class [17] is the family of all $f = x^m + h$ where $\deg(h) \leq m/2$ and $\|h\|_\infty \leq \mathsf{poly}(n)$.

Definition 6 (Middle-Product [23], Definition 3.1). *Let d_a, d_b, d, k be integers such that $d_a + d_b - 1 = d + 2k$. The middle product $\odot_d : R^{<d_a}[x] \times R^{<d_b}[x] \rightarrow R^{<d}[x]$ is defined to be the map: $(a, b) \mapsto a \odot b = \lfloor \frac{(a\cdot b) \mod x^{k+d}}{x^k} \rfloor = \sum_{k\leq i+j\leq k+d-1}(a_ib_j)x^{i+j}$, where $\mathbf{a} = (a_0, \cdots a_{d_a-1})$ and $\mathbf{b} = (b_0, \cdots b_{d_b-1})$ are the coefficient vectors of a and b, respectively.*

Immediately from Definition 6, the middle product is commutative, i.e., $a\odot_d b = b\odot_d a$ for all polynomials a, b. Besides, the middle product also satisfies a "quasi-associative" property defined below in Lemma 4.

Lemma 4 ([23]). *Let $d, k, n > 0$. For all $r \in R^{<k+1}[x], a \in R^{<n}[x], s \in R^{<n+d+k-1}[x]$, we have $r \odot_d (a \odot_{d+k} s) = (r\cdot a) \odot_d s$.*

Lemma 5 ([22], Lemma 3.2). *Let $d, k > 0$. Let $r \in R^{<k+1}[x]$ and $a \in R^{<k+d}[x]$ and $b = r \odot_d a$. Then $\overline{\mathbf{b}} = (\mathsf{T}^{k+1,d}(r))^T \cdot \bar{\mathbf{a}}$. In other words, we have $\mathbf{b} = \overline{(\mathsf{T}^{k+1,d}(r))^T \cdot \mathbf{a}}$.*

Definition 7 (Degree-Parameterized MP-LWE [16], Definition 9). *Let $n > 0$, $q > 2$, $m > 0$, $\mathbf{d} \in [\frac{n}{2}]^t$, and let χ be a distribution over $\mathbb{R}_q$. For $s \in \mathbb{Z}_q^{<n-1}[x]$, we define the distribution $\mathrm{MP}_{q,n,\mathbf{d},\chi}(s)$ over $\Pi_{i=1}^t U(\mathbb{Z}_q^{n-d_i} \times \mathbb{R}_q^{<d_i}[x])$ as follows.*

1. *For each* $i \in [t]$, *sample* $a_i \xleftarrow{\$} \mathbb{Z}_q^{n-d_i}[x]$ *and sample* $e_i \leftarrow \chi^{<d_i}[x]$
2. *Output* $(a_i, b_i := a_i \odot_{d_i} s + e_i)$ *The (degree-parameterized) MP-LWE problem consists of distinguishing between arbitrarily many samples from* $\mathrm{MP}_{q,n,\mathbf{d},\chi}(s)$ *(denoted as* $\mathrm{dMPLWE}_{q,n,\mathbf{d},\chi}(s)$*) and the same number of samples from* $\Pi_{i=1}^{t} U(\mathbb{Z}_q^{n-d_i} \times \mathbb{R}_q^{<d_i}[x])$ *with non-negligible probability over the choice of* $s \xleftarrow{\$} \mathbb{Z}_q^{<n-1}[x]$.

Informally speaking, dMPLWE is as hard as PLWE for a wide class of polynomials. Details can be found in Theorem 2 of [16].

3 Trapdoor Generators and Related Operations

Theorem 1 ([19], Definition 5.2,Theorem 4). *Let* $\mathbf{G} := \mathbf{I}_k \otimes \begin{bmatrix} 1 \ 2 \ \ldots \ 2^{\tau-1} \end{bmatrix} \in \mathbb{Z}_q^{k \times k\tau}$. *Then given a matrix* $\mathbf{A} \in \mathbb{Z}^{k \times (m+k\tau)}$, *we say that a matrix* $\mathbf{R} \in \mathbb{Z}^{m \times k\tau}$ *is a* ***G-trapdoor*** *for* $\mathbf{A}$ *if*

$$\mathbf{A} \left[\mathbf{R}^T | \mathbf{I}_{k\tau} \right]^T = \mathbf{G}. \tag{1}$$

And there exists an efficient algorithm $\mathcal{C}(\mathbf{A}, \mathbf{R}, \mathbf{u})$ *that operates as follows:*

On input matrices $\mathbf{A}, \mathbf{R}$ *and a vector* $\mathbf{u}$, *samples from* $D_{\Lambda_u^{\perp}(\mathbf{A}),\sigma}$, *as long as* $\sigma \geq \omega(\sqrt{\log k})\sqrt{7(\sigma_1(\mathbf{R})^2+1)}$. *Moreover, the running time of* $\mathcal{C}$ *is the time to compute* $\mathbf{Rx}$ *for* $\mathbf{x} \in \mathbb{Z}^{k\tau}$ *plus* $\tilde{O}(m + k\tau)$.

Theorem 2 (Leftover Hash Lemma [16], Theorem 3, Lemma 11). *Let* χ *be a distribution over* $\mathbb{Z}_q$ *and* $\delta \in (0,1)$ *be such that* $H_\infty(\chi) \geq \log(\frac{1}{\delta})$. *Define the distribution* $V := (\overrightarrow{a}, h_{\overrightarrow{a}}(\overrightarrow{r}))$ *over* $S = (\mathbb{Z}_q^{<n}[x])^t \times \mathbb{Z}_q^{<n+n'-1}[x]$, *where* $\overrightarrow{a} = (a_1, \cdots, a_t)$ *consists of i.i.d. samples from* $U(\mathbb{Z}_q^{<n}[x])$, *and* $\overrightarrow{r} = (r_1, \cdots, r_t)$ *consists of i.i.d. samples from* $\chi^{n'}[x]$. *Then for* $n' \leq n$, *if* $\delta^t q = o(1)$, $\Delta(V, U(S)) = O(\delta^{\frac{t}{2}} q + \delta^{\frac{n't}{2}} q^{\frac{n+n'+1}{2}})$. *In particular, for any* $q = \mathsf{poly}(n)$, *if* $\delta^{-1} = \omega(1)$ *and* $n't/n = \Omega(\log n)$, *we have* $V \approx_s U(S)$. *Let* $\chi := D_{\mathbb{Z},\sigma}$ *and* $\chi_q := \chi \mod q$. *For* $\sigma = \mathsf{poly}(n)$, $q = \omega(\sigma \log^{1/2} n)$, $\sigma = \omega(1)$, *we have* $H_\infty(\chi_q) \geq \log(\frac{\sigma}{c})$ *for some constant* c.

Trapdoor Generation and Preimage Sampling Algorithm. We will use the trapdoor generation algorithm and the preimage sampling algorithm to construct our sampling algorithms.

Theorem 3 (TrapGen [16], Theorem 5). *Let* $q = \mathsf{poly}(n)$ *be a prime,* $d \in \mathbb{Z}$, $d \leq n$, $dt/n = \Omega(\log n)$, $\sigma = c\ell \cdot \omega(n \log^{3/2} n)$ *for some constant* c, $\gamma = \lceil \log_2 q \rceil$ *and* $\gamma = \frac{n+2d-2}{d}$ *be an integer. Then there exists a ppt algorithm* $\mathsf{TrapGen}$ *that generates a polynomial vector* $\overrightarrow{a} = (a_1, a_2, \cdots, a_t, a_{t+1}, \cdots, a_{t+\gamma\tau}) \in (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$ *together with a trapdoor* td *that can be stored in* $O(n\tau t)$ *space, where* $\overrightarrow{a} \approx_s U((\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau})$

Theorem 4 (SamplePre($\overrightarrow{a}, \mathsf{td}, u, \sigma$) [16], Theorem 5). *Let $q = \mathsf{poly}(n)$ be a prime, $d \leq n, dt/n = \Omega(\log n), \sigma = c\ell \cdot \omega(n \log^{3/2} n)$ for some constant c, $\gamma = \frac{n+2d-2}{d}$ be an integer, $\tau = \lceil \log_2 q \rceil$ and $t' = t + \gamma\tau$. Then there exists a ppt algorithm* SamplePre *that on input polynomial vector $\overrightarrow{a} = (a_1, a_2, \cdots, a_t, a_{t+1}, \cdots, a_{t+\gamma\tau})$ with a trapdoor* td *and a syndrome $u \in \mathbb{Z}_q^{n+2d-2}[x]$, outputs $\overrightarrow{r} = (r_i)_{i=1}^{t+\gamma\tau}$ satisfying $\sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i = u$ in $\tilde{O}(nt)$ time. And the output distribution of $(r_i)_{i=1}^{t'}$ is exactly the conditional distribution $(D_{\mathbb{Z}^{2d-1},\sigma}[x])^t \times (D_{\mathbb{Z}^d,\sigma}[x])^{\gamma\tau} | \overrightarrow{a} \cdot \overrightarrow{r} = u$.*

After illustrating the algorithms, we would like to propose new algorithms that would be used in our IBE scheme and security proof.

Lemma 6 (SampleLeft($\overrightarrow{a}$, td, u, σ)). *Let $q = \mathsf{poly}(n)$ be a prime, $3d \leq n$, $dt/n = \Omega(\log n)$, $\sigma = c\ell \cdot \omega(n \log^{3/2} n)$ for some constant c, $\gamma = \frac{n+2d-2}{d}$ be an integer, $\tau = \lceil \log_2 q \rceil$ and $t' = t+\gamma\tau$. Then there exists a ppt algorithm* SampleLeft *that on input polynomial vector $\overrightarrow{a} = (a_1, a_2, \cdots, a_t, a_{t+1}, \cdots, a_{t+\gamma\tau})$ with a trapdoor* td*, polynomials $(h_1, h_2, \cdots, h_{\gamma\tau})$, and a syndrome $u \in \mathbb{Z}_q^{n+2d-2}[x]$, outputs $(r_i)_{i=1}^{t+2\gamma\tau}$ satisfying $\sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_i \cdot r_{i+t+\gamma\tau} = u$ in $\tilde{O}(nt)$ time. And the output distribution of (r_i) is exactly the conditional distribution $(D_{\mathbb{Z}^{2d-1},\sigma}[x])^t \times (D_{\mathbb{Z}^d,\sigma}[x])^{2\gamma\tau} | \sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_i \cdot r_{i+t'} = u$.*

Proof. Let $t' := t + \gamma\tau$ and $t'' := t + 2\gamma\tau$.
First, we describe SampleLeft algorithm and then prove that it outputs the right distribution of polynomials (r_i):

Chose i.i.d. samples $r_{t'+1}, r_{t'+2}, ..., r_{t''} \leftarrow (D_{\mathbb{Z}^d,\sigma}[x])^t \times (D_{\mathbb{Z}^d,\sigma}[x])^{\gamma\tau}$. Then set $(r_1, r_2, ..., r_{t'}) \leftarrow \mathsf{SamplePre}((a_1, a_2, ..., a_t, a_{t+1}, ..., a_{t+\gamma\tau}), \mathsf{td}, u - \sum_{i=1}^{\gamma\tau} h_i \cdot r_{i+t'}, \sigma)$. We can see that $\sum_{i=1}^{t'} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_i \cdot r_{i+t'} = u$.

Next, we are to prove that the SampleLeft algorithm outputs the right distribution of polynomials (r_i).

First, according to the algorithm, for every $u \in \mathbb{Z}_q^{n+2d-2}[x]$, we can find polynomials $(D_{\mathbb{Z}^{2d-1},\sigma}[x])^t \times (D_{\mathbb{Z}^d,\sigma}[x])^{2\gamma\tau} s.t. \sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} a_i \cdot r_{i+t'} = u$.

Let $u' = u - \sum_{i=1}^{\gamma\tau} a_i \cdot r_{i+t'}$. According to the Leftover Hash Lemma (Theorem 2), the distribution of u' is also $U(\mathbb{Z}_q^{<n+2d-2})$. We have $(r_1, ..., r_{t'})$ generated using $\mathsf{SamplePre}((a_1, a_2, ..., a_t, a_{t+1}, ..., a_{t+\gamma\tau}), \mathsf{td}, u', \sigma)$.

According to Theorem 1, we can see that the distribution of $(r_i)_{i=1}^{t'}$ is exactly the conditional distribution $(D_{\mathbb{Z}^{2d-1},\sigma}[x])^t \times (D_{\mathbb{Z}^d,\sigma}[x])^{\gamma\tau} | \sum_{i=1}^{t'} a_i \cdot r_i = u'$. Since $(r_i)_{i=t'+1}^{t''}$ are sampled i.i.d. from $(D_{\mathbb{Z}^d,\sigma}[x])^{\gamma\tau}$, we can come to the conclusion that $(D_{\mathbb{Z}^{2d-1},\sigma}[x])^t \times (D_{\mathbb{Z}^d,\sigma}[x])^{2\gamma\tau} | \sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_i \cdot r_{t'+\gamma\tau} = u$. Thus, the conditional distribution of (r_i) is as claimed.

Finally, the runtime of SampleLeft is precisely the runtime of SamplePre, which is $\tilde{O}(nt)$ time. □

Lemma 7 (SampleRight($\overrightarrow{b}, y, \overrightarrow{a}, \overrightarrow{S}, u, \sigma$)). *Let $q = \mathsf{poly}(n)$ be a prime, $3d \leq n, dt/n = \Omega(\log n), \sigma = c\ell \cdot \omega(n \log^{3/2} n)$ for some constant c, $\gamma = \frac{n+2d-2}{d}$ is an integer, $\tau = \lceil \log_2 q \rceil$ and $t' = t + \gamma\tau$. Then there exists a ppt algorithm*

SampleRight *that on input polynomial vector* $\overrightarrow{b} \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, *a non-zero integer* $y \in \mathbb{Z}^*$, *polynomial vector* $\overrightarrow{a} = (a_1, a_2, \cdots, a_{t+\gamma\tau})$, *and polynomial matrix* $\overrightarrow{S} = (\overrightarrow{s_1}, \overrightarrow{s_2}, \cdots, \overrightarrow{s_{t'}})^T$ *where* $\overrightarrow{s_j} \in (\mathbb{Z}_q^d)^t \times (\mathbb{Z}_q)^{\gamma\tau}$ *for* $1 \leq j \leq \gamma\tau$ *and a syndrome* $u \in \mathbb{Z}_q^{n+2d-2}[x]$, *outputs polynomial vector* $\overrightarrow{r} = (r_i)_{i=1}^{t+2\gamma\tau}$ *satisfying* $\sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_i \cdot r_{t+i+\gamma\tau} = u$, *where* $h_i = \overrightarrow{s_i} \cdot \overrightarrow{a} + \overrightarrow{b}$ *for* $i \in [\gamma\tau]$ *in* $\tilde{O}(nt)$ *time. And the output distribution of* (r_i) *is exactly the conditional distribution* $(D_{\mathbb{Z}^{2d-1},\sigma}[x])^t \times (D_{\mathbb{Z}^d,\sigma}[x])^{2\gamma\tau} | \sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_i \cdot r_{i+t+\gamma\tau} = u'$.

Proof. The proof consists of four parts. In the first part, we give a sampling algorithm SampleRight originated from the sampling algorithm proposed by [19]. In the second part, we prove the correctness of our SampleRight algorithm. In the third part, we prove that the output of our algorithm follows the desired distribution. Finally, we analyze the running time of our algorithm. Our sampling algorithm is as follows. Let $u' = y^{-1}u$. Here y^{-1} denotes the multiplicative inverse of y in $\mathbb{Z}_q$. Set $\mathbf{\Gamma}(f) = [T^{n+d-1,d}(f)|\dots|T^{n+d-1,d}(f2^{\tau-1})]$, $\mathbf{G} = \left[\mathbf{\Gamma}(1)|\mathbf{\Gamma}(x^d)|\dots|\mathbf{\Gamma}(x^{(\gamma-1)d})\right]$.

Let $\tilde{\mathbf{A}} = \left[\mathsf{T}^{n,2d-1}(a_1)|\dots|\mathsf{T}^{n,2d-1}(a_t)|\mathsf{T}^{n+d-1,d}(a_{t+1})|\dots|\mathsf{T}^{n+d-1,d}(a_{t+\gamma\tau})\right]$,

$$\tilde{\mathbf{S}} = \left[\begin{array}{ccc} \mathsf{T}^{d,d}(s_{1,1}) & \dots & \mathsf{T}^{d,d}(s_{1,\gamma\tau}) \\ \vdots & \vdots & \vdots \\ \mathsf{T}^{d,d}(s_{t,1}) & \dots & \mathsf{T}^{d,d}(s_{t,\gamma\tau}) \\ \hline \mathsf{T}^{1,d}(s_{t+1,t+1}) & \dots & \mathsf{T}^{1,d}(s_{t+1,\gamma\tau}) \\ \vdots & \vdots & \vdots \\ \mathsf{T}^{1,d}(s_{t',t+1}) & \dots & \mathsf{T}^{1,d}(s_{t',\gamma\tau}) \end{array}\right] \text{ and } \mathbf{I} = \mathbf{I}_{d\gamma\tau} = \begin{bmatrix} \mathsf{T}^{1,d}(1) & \dots & \mathsf{T}^{1,d}(1) \\ \vdots & \ddots & \vdots \\ \mathsf{T}^{1,d}(1) & \dots & \mathsf{T}^{1,d}(1) \end{bmatrix}.$$

We have $\mathbf{G} = \mathbf{I}_{\gamma d} \otimes [1 \cdots 2^{\tau-1}] = [\mathsf{T}^{n+d-1,d}(b_1)| \cdots |\mathsf{T}^{n+d-1,d}(b_{\gamma\tau})]$. Set $\mathbf{A} = [\tilde{\mathbf{A}}|\tilde{\mathbf{A}}\tilde{\mathbf{S}} + \mathbf{G}]$ and $\mathbf{S} = [-\tilde{\mathbf{S}}^T|\mathbf{I}]^T$. We can get $\mathbf{AS} = \mathbf{G}$

Here, $\tilde{\mathbf{A}}$ is the Toepliz matrix corresponding to $\overrightarrow{a}$, $\mathbf{A}$ is the Toepliz matrix corresponding to $[\overrightarrow{a}^T|\overrightarrow{h}^T]^T$ and $-\tilde{\mathbf{S}}$ is the $\mathbf{G}$-trapdoor for $\mathbf{A}$.

Let $\mathbf{u}' = \mathsf{T}^{n+2d-2,1}(u') \in \mathbb{Z}_q^{n+2d-2}$ be the coefficient vector of u'.

After that, apply the algorithm from Theorem 1 and set $k = \gamma d = n+2d-2$, we can sample $\mathbf{e}$ from $D_{\Lambda_{\mathbf{u}'}^{\perp}(\mathbf{A}),\sigma}$, where $\sigma = c\ell \cdot \omega(n \log^{3/2} n) \geq \omega(\sqrt{\log(\gamma d)}) \cdot \sqrt{7((\gamma d\tau((2d-1)t + d\gamma\tau)) \cdot \ell^2 + 1)} = \omega(\sqrt{\log \gamma d})\sqrt{7(\sigma_1(\tilde{S})^2) + 1}$ for some constant c.

Next, we separate $\mathbf{e}$ into a series of Toeplize matrices, i.e., write $\mathbf{e}$ as $[\mathsf{T}^{2d-1,1}(r_1)^T| \cdots |\mathsf{T}^{2d-1,1}(r_t)^T|\mathsf{T}^{d,1}(r_{t+1})^T| \cdots |\mathsf{T}^{d,1}(r_{t+2\gamma\tau})^T]^T$,

where $\deg(r_i) = \begin{cases} < 2d-1 & \text{for } i \in [t] \\ < d & \text{for } i \in \{t+1, \cdots, t+2\gamma\tau\}. \end{cases}$

Finally, output$(r_1, \cdots, r_{t+2\gamma\tau})$.

In the second part of the proof, we analyze the correctness of SampleRight. By construction, we have $\max_{i,j}|\tilde{S}_{i,j}| \leq \ell$. So, according to Lemma 1 and Theorem 1, we can get $\sigma_1(\tilde{\mathbf{S}}) \leq \ell\sqrt{\gamma d\tau((2d-1)t + d\gamma\tau)}$ and $\mathbf{e}$ is sampled from $D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}),\sigma}$. Here, Theorem 1 works as $k = \gamma d = n+2d-2 \leq 3n$ and $\gamma = \Theta(\log q) = \Theta(\log n)$ for $q = \mathsf{poly}(n)$.

Then, as we have

$$\mathbf{A} = \begin{bmatrix} \mathsf{T}^{n,2d-1}(a_1)|\dots|\mathsf{T}^{n,2d-1}(a_t)|\mathsf{T}^{n+d-1,d}(a_{t+1})|\dots|\mathsf{T}^{n+d-1,d}(a_{t+\gamma\tau}) \\ \dots|\mathsf{T}^{n+d-1,d}(h_1)|\dots|\mathsf{T}^{n+d-1,d}(h_{\gamma\tau}) \end{bmatrix},$$

by Lemma 2, we can deduce that $\mathbf{Ae} = \mathsf{T}^{n+2d-2,1}(\sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_i \cdot r_{i+t+\gamma\tau})$. Therefore, $\mathbf{e} \in \Lambda_{\mathbf{u}'}^{\perp}(\mathbf{A})$ iff $\sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_i \cdot r_{i+t'} = u'$.

Thirdly, we prove that $r = (r_i)_{i=1}^{t'+\gamma\tau}$ is distributed as claimed. Since $\mathbf{AS} = \mathbf{G}$ and $\text{span}(\mathbf{G}) = \mathbb{Z}^{n+2d-2}$, we can get $\text{span}(\mathbf{A}) = \mathbb{Z}^{n+2d-2}$. So $\forall \mathbf{u}' \in \mathbb{Z}^{n+2d-2}$, $\exists \mathbf{e}^*$ s.t. $\mathbf{Ae}^* = \mathbf{u}'$. Since $\forall \mathbf{t}$ s.t. $\mathbf{At} = \mathbf{u}'$, $\mathbf{t} + \Lambda^{\perp}(\mathbf{A}) = \Lambda_{\mathbf{u}'}^{\perp}(\mathbf{A})$ and $D_{\Lambda_{\mathbf{u}'}^{\perp}(\mathbf{A}),\sigma}(\mathbf{x}) = \rho_\sigma(\mathbf{x})/\rho_\sigma(\Lambda_{\mathbf{u}'}^{\perp}(\mathbf{A})) = D_{\Lambda_{\mathbf{u}'}^{\perp}(\mathbf{A}),\sigma,-\mathbf{t}}(\mathbf{x} - \mathbf{t}), \forall \mathbf{x} \in \Lambda_{\mathbf{u}'}^{\perp}(\mathbf{A})$, we can conclude that $D_{\Lambda_{\mathbf{u}'}^{\perp}(\mathbf{A}),\sigma} \equiv \mathbf{e}^* + D_{\Lambda^{\perp}(\mathbf{A}),\sigma,-\mathbf{e}^*}$.

Then, by Lemma 5.2 in [12], the distribution of $\mathbf{e}$ is as follows: $D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}),\sigma} \equiv \mathbf{e}^* + D_{\Lambda^{\perp}(\mathbf{A}),\sigma,-\mathbf{e}^*} \equiv D_{\mathbb{Z},\sigma}^{(2d-1)t+2d\gamma\tau}|\mathbf{Ae} = \mathbf{u}'$. So the conditional distribution of $\overrightarrow{r}$ is as claimed.

Finally, we analyze the running time of the $\mathsf{SamplePre}$ algorithm. The running time of $\mathsf{SampleRight}$ is the time needed to calculate $\tilde{\mathbf{R}}\mathbf{x}$ for $\mathbf{x} \in \mathbb{Z}_q^{\gamma d\tau}$ using polynomial multiplication, which is $O((d \log d)t\gamma\tau) = \tilde{O}(nt)$, as $\gamma d = O(n)$, $\log d = O(\log n)$ and $\gamma = \Theta(n)$. □

4 Our Middle-Product IBE

Overview of Our Construction. In our scheme, a user's identity $\mathsf{id} = \{-1, 1\}^\ell$ corresponds to a polynomial vector $\overrightarrow{h_{\mathsf{id}}} \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, which is generated as follows: $\overrightarrow{h_{\mathsf{id}}} = \overrightarrow{h_0} + \sum_{j=1}^{\ell} d_j \overrightarrow{h_j}$, where d_j is the jth bit of the identity id and $\overrightarrow{h_0}, \overrightarrow{h_1}, \cdots, \overrightarrow{h_\ell} \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$ are polynomial vectors included in the master public key. The master public key also contains a polynomial u, and a polynomial vector $\overrightarrow{a} \in (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$. Here $\overrightarrow{a}$ along with its trapdoor td are generated by $\mathsf{TrapGen}$ algorithm introduced in Theorem 3. Set $t' = t + \gamma\tau$ and $t'' = t' + \gamma\tau$. The private key of a user id is a sequence of polynomials $(r_1, r_2, \cdots, r_{t''})$ satisfying $\sum_{i=1}^{t'} a_i \cdot r_i + \sum_{i=1}^{\gamma\tau} h_{\mathsf{id},i} \cdot r_{i+t'} = u$, which can be sampled by $\mathsf{SampleLeft}$ algorithm defined in Lemma 6 using trapdoor td.

Let $q = q(n)$ be prime, $\tau := \lceil log_2 q \rceil$, n, d, k be such that $\gamma = \frac{n+2d-2}{d} \in \mathbb{N}$ and $2d + k \leq n$. Let $t > 0$, $t' = t + \gamma\tau$, $t'' = t + \gamma\tau$. Let $\chi := \lfloor D_{\alpha \cdot q} \rceil, \chi_1 := \lfloor D_{\alpha_1 \cdot q} \rceil$ be the distributions over $\mathbb{Z}$ in which $\epsilon \leftarrow D_{\alpha \cdot q}$, or $\epsilon_1 \leftarrow D_{\alpha_1 \cdot q}$ is sampled and then rounded to the nearest integer respectively. Let $0 < s \leq d/2$ be an integer. Let identity $\mathsf{id} = \{-1, 1\}^\ell$, where ℓ is the length of the identity id. Let $\sigma = c\ell \cdot \omega(n \log^{3/2} n)$ for some constant c. We define an IBE scheme with message space $\mathcal{M} = \{0, 1\}^{<k+1}[x]$.

Key Generation $\mathsf{KeyGen}(1^n)$:

1. Use $\mathsf{TrapGen}$ to generate random polynomials $(a_1, \dots, a_t, a_{t+1}, \dots, a_{t'}) \leftarrow (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$ with trapdoor td, and denote it as $\overrightarrow{a}$.

2. Sample $\overrightarrow{h_0}, \overrightarrow{h_1}, \ldots, \overrightarrow{h_\ell} \xleftarrow{\$} (\mathbb{Z}_q^{<n+d-1})^{\gamma\tau}$
3. Sample $u \xleftarrow{\$} \mathbb{Z}_q^{n+2d-2}$.
4. Output $\mathsf{mpk} = \{\overrightarrow{a}, \overrightarrow{h_0}, \overrightarrow{h_1}, \ldots, \overrightarrow{h_\ell}, u\}, \mathsf{msk} = \mathsf{td}$.

Extraction $\mathsf{Extract}(\mathsf{mpk}, \mathsf{msk}, \mathsf{id})$:

1. Let $\overrightarrow{h_{\mathsf{id}}} = \overrightarrow{h_0} + \sum_{j=1}^{\ell} d_j \overrightarrow{h_j}$, where d_j is the jth bit of the identity id.
2. Use $\mathsf{SampleLeft}(\overrightarrow{a}$, $\overrightarrow{h_{\mathsf{id}}}$, td, u, $\sigma)$ to generate $(r_1, \ldots, r_{t''})$ s.t. $\sum_{i=0}^{t'} a_i\, r_i + \sum_{i=1}^{\gamma\tau} h_{\mathsf{id},i}\, r_{i+t'} = u$.
3. Output $\mathsf{sk}_{\mathsf{id}} = (r_1, \cdots, r_{t''})$

Encryption $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$:

1. Let $\overrightarrow{h_{\mathsf{id}}} = \overrightarrow{h_0} + \sum_{j=1}^{\ell} d_j \overrightarrow{h_j}$, where d_j is the jth bit of the identity id.
2. Sample $p \xleftarrow{\$} \mathbb{Z}_q^{<n+2d+k-1}[x]$.
3. For $1 \le i \le t$, sample $e_i \leftarrow \chi^{2d+k}[x]$, and compute $c_i = a_i \odot_{2d+k} p + 2e_i$.
4. For $t+1 \le i \le t'$, sample $e_i \leftarrow \chi^{d+k+1}[x]$, and compute $c_i = a_i \odot_{d+k+1} p + 2e_i$.
5. For $1 \le i \le \gamma\tau$, sample $e_{i+t'} \leftarrow \chi_1^{d+k+1}[x]$, and compute $c_{i+t'} = h_{\mathsf{id},i} \odot_{d+k+1} p + 2e_{i+t'}$.
6. Sample $e_0 \leftarrow \chi^{k+1}$, and compute $c_0 = m + u \odot_{k+1} p + 2e_0$.
7. Output $c = (c_0, c_1, \cdots, c_{t''})$.

Decryption $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c)$: Output $(c_0 - \sum_{i=1}^{t''} c_i \odot_{k+1} r_i \mod q) \mod 2$

Lemma 8. *For* $\alpha_1 = (n^2\ell \cdot \omega(\log^{\frac{7}{2}} n) + 1)^{-1}$, *and* $\alpha = (n^3\ell^2 \cdot \omega(\log^4 n) + 1)^{-1}$, *the scheme satisfies* $(1 - \mathsf{negl}(n))$*-correctness.*

Proof. We want to show that $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, \mu)) = \mu$ with probability $1 - \mathsf{negl}(n)$ over the randomness of KeyGen and Enc. Consider a random key pair $(\mathsf{mpk}, \mathsf{msk})$, a random identity $\mathsf{id} = \{-1, 1\}^l$, let $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{Extract}(\mathsf{mpk}, \mathsf{msk}, \mathsf{id})$ and compute ciphertext $c = (c_0, (c_i)_{1 \le i \le t''}) \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$.

By Lemma 4 (the quasi-associative law for middle products), we have $c_0 = m + (\sum_{i=0}^{t'} a_i r_i + \sum_{i=1}^{\gamma\tau} h_{\mathsf{id},i} r_{i+t'}) \odot_{k+1} p + 2e' = m + \sum_{i=1}^{t} r_i \odot_{k+1} (a_i \odot_{2d+k} p) + \sum_{i=t+1}^{t'} r_i \odot_{k+1} (a_i \odot_{d+k+1} p) + \sum_{i=1}^{\gamma\tau} r_{i+t'} \odot_{k+1} (h_{\mathsf{id},i} \odot_{d+k+1} p) + 2e'$, which leads to the following equation $c_0 - \sum_{i=1}^{t''} c_i \odot_{k+1} r_i = m + 2(e_0 - \sum_{i=1}^{t''} r_i \odot_{k+1} e_i)$. So if $\|m + 2(e_0 - \sum_{i=1}^{t''} r_i \odot_{k+1} e_i)\|_\infty < q/2$, $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c)$ will output the message m correctly.

Hence, to fulfill the proof of correctness, we are to bound the coefficients of $\sum_{i=1}^{t''} r_i \odot_{k+1} e_i$.

The coefficient of x^l in $r_i \odot_{k+1} e_i$ is $\sum_{\omega \in [0, deg(r_i)] \cap [l+k-deg(e_i), z+k]} (\mathbf{y}_i)_\omega$ $(\mathbf{e}_i)_{l+k-\omega}$. Applying the Gaussian tail inequality and a union bound, since $\alpha_1 > \alpha$, we can get the following bounds on $\|\mathbf{y}_i\|_\infty$ and $\|\mathbf{e}_i\|_\infty$: $\Pr[\|\mathbf{y}_i\|_\infty > \omega((\sqrt{\log n})\sigma)] = \mathsf{negl}(n)$. $\Pr[\|\mathbf{e}_i\|_\infty > \omega((\sqrt{\log n})\alpha_1 q)] = \mathsf{negl}(n)$.

Therefore, again by union bound, except with $\mathsf{negl}(n)$ probability $\|e_0 - \sum_{i=1}^{t''} r_i \odot_{k+1} e_i\|_\infty < K\omega((\sqrt{\log n})\sigma)\omega((\sqrt{\log n})\alpha_1 q) + \omega((\sqrt{\log n})\alpha_1 q)$ for $K := (2d-1)t + 2\gamma\tau d \geq \sum_{i=1}^{t''}(deg(r_i)+1)$.

Setting $\alpha_1 < (4\sigma K \cdot \omega(\log n)+1)^{-1} = (n^2\ell \cdot \omega(\log^{\frac{7}{2}} n)+1)^{-1}$, the above is less than $q/4$ and thus the scheme is $(1-\mathsf{negl}(n))$-correct. In this case, $\alpha = (n^3 l^2 \cdot \omega(\log^4 n)+1)^{-1}$. □

4.1 Security Proof

In this section, we will prove the security of our IBE scheme. We adapt the partitioning technique, following the previous works [1,4,26,28,29].

Overview of Our Proof. In the proof, we will first change the way of generating master public keys. Instead of using $\mathsf{TrapGen}$ algorithm to generate $\overrightarrow{a}$, we sample $\overrightarrow{a} = (a_1, \cdots, a_t, a_{t+1}, \cdots, a_{t'}) \xleftarrow{\$} (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$. We also sample polynomial matrices $\overrightarrow{S_0}, \overrightarrow{S_1}, \cdots, \overrightarrow{S_\ell}$ where $\overrightarrow{S_i} = [\overrightarrow{s_{i,1}}, \overrightarrow{s_{i,2}}, \cdots, \overrightarrow{s_{i,t'}}]^T$ and $\overrightarrow{s_{i,j}} \leftarrow (\Psi^d[x])^t \times \Psi^{\gamma\tau}$ for $\Psi := U(\{-1,0,1\})$, $j = 1, \cdots, t'$, along with $\mathbf{y} = (y_1, \cdots, y_\ell) \in \mathbb{Z}_q^\ell$ and $y_0 = 1$. After that, we will set the remaining part of master public key as $(\overrightarrow{h_i})_{i=0}^\ell$ where $\overrightarrow{h_i} = \overrightarrow{S_i} \cdot \overrightarrow{a} + y_i \overrightarrow{b}$. In this way $\overrightarrow{h_{\mathsf{id}}}$ can be written as $(\overrightarrow{S_0} + \sum_{i=0}^\ell d_i \overrightarrow{S_i}) \cdot \overrightarrow{a} + (1 + \sum_{i=1}^\ell d_i y_i)\overrightarrow{b}$. Then an abort-resistant hash function can be defined as follows: $F_{\mathbf{y}}(\mathsf{id}) = y_0 + \sum_{i=1}^\ell d_i y_i$. We also define $\overrightarrow{S_{\mathsf{id}}} = \overrightarrow{S_0} + \sum_{i=0}^\ell d_i \overrightarrow{S_i}$. For identity queries $\{\mathsf{id}_1, \cdots, \mathsf{id}_Q\}$ and challenge identity id^*, we further define $\Upsilon(\mathsf{id}^*, \mathsf{id}_1, \cdots, \mathsf{id}_Q)$ as the probability that all key extraction queries have $F_{\mathbf{y}}(\mathsf{id}_i) \neq 0$ and challenge identity has $F_{\mathbf{y}}(\mathsf{id}^*) = 0$. With artificial abort, we are able to argue that we have $\Upsilon(\mathsf{id}^*, \mathsf{id}_1, \cdots, \mathsf{id}_Q)$ non-negligible for all possible identity queries. In our setting $\overrightarrow{b}$ is generated with a trapdoor. Then we transform the polynomial vectors $\overrightarrow{a}$, $\overrightarrow{b}$ and $\overrightarrow{S_{\mathsf{id}}}$ to matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{S}$ respectively. In the key extraction queries, we are to find $\mathbf{e}$, the vector representation of user secret key s.t. $[\mathbf{A}|\mathbf{SA} + F_{\mathbf{y}}(\mathsf{id})\mathbf{B}]\mathbf{e} = \mathbf{u}$ for the predetermined $\mathbf{u}$ in master public key. As $[\mathbf{A}|\mathbf{SA} + F_{\mathbf{y}}(\mathsf{id})\mathbf{B}]\left[-\mathbf{S}^T|\mathbf{I}\right]^T = F_{\mathbf{y}}(\mathsf{id})\mathbf{B}$, when $F_{\mathbf{y}}(\mathsf{id}) \neq 0$, we can sample $\mathbf{e}$ using the trapdoor in $\mathbf{B}$ by the $\mathsf{SampleRight}$ algorithm. This allows us to answer the key extraction queries. For the challenge identity id^*, we have $F_{\mathbf{y}}(\mathsf{id}^*) = 0$ and $\overrightarrow{h_{\mathsf{id}}} = \overrightarrow{S_{\mathsf{id}}} \cdot \overrightarrow{a}$. We will use MPLWE samples as ciphertext $(c_i)_{i=1}^{t'}$ and $c_0 - m_b$ where m_b is the challenge message. We will then generate the remaining ciphertext $(c_i)_{i=t'+1}^{t''}$ from $(c_i)_{i=1}^{t'}$ using the rerandomization algorithm ReRand from [14]. The rest of the analysis is straightforward: the ciphertext contains no information about b under the MPLWE assumption.

Our IBE is IND-CPA secure with the parameter setting given below.

Parameter Settings. The parameters used in our IBE are listed below.

$$d = \Theta(n) \qquad t = \Omega(\log n) \qquad q = n^{4+c}\ell^2 \cdot \omega(\log^4 n)$$
$$\sigma = \ell \cdot \omega(n \log^{\frac{3}{2}} n) \; \alpha = (n^3\ell^2 \cdot \omega(\log^4 n)+1)^{-1} \; \alpha_1 = (n^2\ell \cdot \omega(\log^{\frac{7}{2}} n)+1)^{-1}$$

Definition 8 (Abort-resistant Hash Function [1]**).** *Let $\mathcal{H} := \{H : X \to Y\}$ be a family of hash functions from X to Y where $0 \in Y$. For a set of $Q+1$ in puts $\bar{x} = (x_0, x_1, \cdots, x_Q) \in X^{Q+1}$, define the non-abort probability of $\bar{x}$ as the quantity $\alpha(\bar{x}) := \Pr[H(x_0) = 0 \wedge H(x_1) \neq 0 \wedge \cdots \wedge H(x_Q) \neq 0]$ where the probability is over the random choice of $H \in \mathcal{H}$.*

We say that $\mathcal{F}$ is $(Q, \alpha_{\min}, \alpha_{\max})$ abort-resistant if for all $\bar{x} = (x_0, x_1, \cdots, x_Q) \in X^{Q+1}$ with $x_0 \notin \{x_1, \cdots, x_Q\}$, we have $\alpha(\bar{x}) \in [\alpha_{\min}, \alpha_{\max}]$.

We will use the following abort-resistant hash function used in [1,4,13,26]. And this is the main component that enables us to go from the selectively secure IBE in [16] to an adaptively secure one.

For a prime q, let $(\mathbb{Z}_q^\ell)^* := \mathbb{Z}_q^\ell \setminus \{0^\ell\}$ and define the family $\mathcal{F}_{\mathrm{Wat}} : \{F_{\mathbf{y}} : (\mathbb{Z}_q^l)^* \to \mathbb{Z}_q\}_{\mathbf{y} \in \mathbb{Z}_q^\ell}$ as $F_{\mathbf{y}}(\mathsf{id}) := 1 + \sum_{i=1}^{\ell} y_i b_i \in \mathbb{Z}_q$ where $\mathsf{id} = (b_1, \cdots b_\ell) \in (\mathbb{Z}_q^\ell)^*$ and $\mathbf{y} = (y_1, \cdots, y_\ell) \in \mathbb{Z}_q^\ell$

In our IBE scheme, the inputs to these hash functions are in $\{-1, 1\}^\ell$. Since $\{-1, 1\}^\ell \in (\mathbb{Z}_q^\ell)^* := \mathbb{Z}_q^\ell \setminus \{0^\ell\}$, we will use the more general result in [1].

Lemma 9 ([1])**.** *Let q be a prime and $0 < Q < q$. Then the hash family $\mathcal{F}_{\mathrm{Wat}}$ defined above is $(Q, \frac{1}{q}(1 - \frac{Q}{q}), \frac{1}{q})$ abort-resistant.*

Theorem 5. *Assume that $\sigma = c\ell \cdot \omega(n \log^{3/2} n)$ for some constant c, $dt/n = \Omega(\log n)$, $q = \mathsf{poly}(n)$ is a prime number, $q = \Omega(\alpha^{-1} n^{1+c})$ and $q = \sigma \cdot \omega(\log^{1/2} n)$, $\alpha_1 < (4\sigma K \cdot \omega(\log n) + 1)^{-1}$, and $\alpha = \alpha_1/(2\sqrt{t'}(2d+k)l)$ where $K := (2d-1)t + 2\gamma\tau d$, the IBE system is IND-CPA adaptively secure assuming* $\mathrm{dMPLWE}_{q,n+2d+k,\mathbf{d},D_{\alpha \cdot q}}$ *is hard with degree vector $\mathbf{d} = (d_i)_{i=1}^{t'+1}$ where*

$$d_i = \begin{cases} 2d+k, \mathit{for}\ 1 \leq i \leq t \\ d+k+1, \mathit{for}\ t+1 \leq i \leq t' \\ k+1, \mathit{for}\ i = t''+1 \end{cases}$$

Proof. The proof proceeds in a sequence of games, among which the first game is identical to the IND-CPA adaptive game from Sect. 2.2. In the last game in the sequence, the adversary has an advantage of zero. In Game 1, we use an artificial abort to check if identities $(\mathsf{id}^*, \mathsf{id}_1, \cdots, \mathsf{id}_Q)$ selected by the challenger $\mathcal{C}$ satisfy our requirement where id^* is the challenge identity and $(\mathsf{id}_1, \cdots \mathsf{id}_Q)$ are for key extraction queries. Then we change the generation of mpk such that for $\mathsf{id}_1, \cdots, \mathsf{id}_Q$, we are able to extract the secret key using embedded trapdoor, while for challenge id^* there is no such trapdoor available for key extraction. Then the dMPLWE problem is used in proving that Game 6 and Game 7 are indistinguishable.
In each game, two values $\mathsf{coin}', \widetilde{\mathsf{coin}} \in \{0, 1\}$ are defined. While we set $\mathsf{coin}' = \widetilde{\mathsf{coin}}$ in the first game, these values might be different in the later guess when artificial abort occurs. Finally, we will define X_i be the event that $\mathsf{coin}' = \mathsf{coin}$.

Game 0. This is the original adaptive IND-CPA security game from Sect. 2.2, between an PPT attacker $\mathcal{A}$ and a challenger $\mathcal{C}$.

Recall that the ciphertext space is $c = (c_0, (c_i)_{i \le t''}) = \mathbb{Z}_q^{k+1}[x] \times ((\mathbb{Z}_q^{2d+k}[x])^t \times (\mathbb{Z}_q^{d+k+1}[x])^{2\gamma\tau})$. In the challenge phase, the challenge ciphertext is generated using our IBE scheme by encrypting message m_{coin}. At the end of the game, $\mathcal{A}$ outputs a guess $\widetilde{\mathsf{coin}}$ for coin. Finally, $\mathcal{C}$ sets $\mathsf{coin}' = \widetilde{\mathsf{coin}}$. By definition, we have $|\Pr[X_0] - \frac{1}{2}| = |\Pr[\mathsf{coin}' = \mathsf{coin}] - \frac{1}{2}| = \epsilon$.

Game 1. Game 1 is identical to Game 0 except that we add an abort event at the beginning of the game.

The challenger $\mathcal{C}$ picks $\mathbf{y} = (y_1, \cdots, y_\ell) \in \mathbb{Z}_q^\ell$. We define a function $F_\mathbf{y} : \mathcal{ID} \mapsto \mathbb{Z}_q$ as follows: $F_\mathbf{y}(\mathsf{id}) = y_0 + \sum_{i=1}^{\ell} d_i y_i$, where d_i is the ith bit of identity id and $\mathcal{ID} \in (\mathbb{Z}_q^\ell)^*$. Then $\mathcal{C}$ checks whether the following condition holds:

$$F_\mathbf{y}(\mathsf{id}^*) = 0 \wedge F_\mathbf{y}(\mathsf{id}_1) \neq 0 \wedge F_\mathbf{y}(\mathsf{id}_2) \neq 0 \wedge \cdots \wedge F_\mathbf{y}(\mathsf{id}_Q) \neq 0 \qquad (2)$$

where id^* is the challenge identity and $\mathsf{id}_1, \cdots \mathsf{id}_Q$ are the identities for which $\mathcal{A}$ has made key extraction queries. And set $\varUpsilon(\mathsf{id}^*, \mathsf{id}_1, \cdots, \mathsf{id}_Q) = \Pr_\mathbf{y}[F_\mathbf{y}(\mathsf{id}^*) = 0 \wedge F_\mathbf{y}(\mathsf{id}_1) \neq 0 \wedge F_\mathbf{y}(\mathsf{id}_2) \neq 0 \wedge \cdots \wedge F_\mathbf{y}(\mathsf{id}_Q) \neq 0]$. Here $F_\mathbf{y}(\mathsf{id})$ is an abort-resistant hash function.

Now $\mathcal{C}$ does as follows:

(I) Abort check: If (2) does not hold, $\mathcal{C}$ ignores the output $\widetilde{\mathsf{coin}}$ of $\mathcal{A}$, and set $\mathsf{coin}' \xleftarrow{\$} \{0, 1\}$. In this case, we say that the challenger aborts. If condition (2) holds, the challenger $\mathcal{C}$ sets $\mathsf{coin}' = \widetilde{\mathsf{coin}}$.

(II) Artificial abort: The artificial abort does in the same way as in [1]. With probability $\varUpsilon(\mathsf{id}^*, \mathsf{id}_1, \cdots, \mathsf{id}_Q)$, $\mathcal{C}$ ignores the output $\widetilde{\mathsf{coin}}$ of $\mathcal{A}$, and sets $\mathsf{coin}' \xleftarrow{\$} \{0, 1\}$, otherwise, $\mathcal{C}$ sets $\mathsf{coin}' = \widetilde{\mathsf{coin}}$. Note that the abort condition is determined using the hash function $F_\mathbf{y}(\cdot)$ that is independent of the attacker's view. For a $(Q+1)$-tuple of identities $I = (\mathsf{id}^*, \mathsf{id}_1, \mathsf{id}_2, \cdots, \mathsf{id}_Q)$, denotes the probability that an abort (either real or artificial abort) does not happen when $\mathcal{A}$ makes these queries by $\varUpsilon(I)$. Let $\varUpsilon_{\max}$ and $\varUpsilon_{\min}$ be such that $\varUpsilon(I) \in [\varUpsilon_{\min}, \varUpsilon_{\max}]$ for all $(Q+1)$ tuples of identities I.

Lemma 10. ([1], Lemma 28). *For $i = 0, 1$, let X_i be the event that $\mathsf{coin}' = \mathsf{coin}$ at the end of Game i. With the artificial abort,* $|\Pr[X_1] - \frac{1}{2}| \ge \varUpsilon_{\min}|\Pr[X_0] - \frac{1}{2}| - \frac{1}{2}(\varUpsilon_{\max} - \varUpsilon_{\min})$ *and* $(\varUpsilon_{\max} - \varUpsilon_{\min}) \le \alpha_{\min}|\Pr[X_0] - 1/2|$

So we can get $|\Pr[X_1] - \frac{1}{2}| \ge \frac{1}{2} \cdot \alpha_{\min}|\Pr[X_0] - \frac{1}{2}| \ge \frac{1}{4q}|\Pr[X_0] - \frac{1}{2}|$.

Game 2. In Game 2, we slightly change the way that $\mathcal{C}$ generates the polynomial vectors $\overrightarrow{h_0}, \overrightarrow{h_1}, \cdots, \overrightarrow{h_\ell}$. Instead of selecting them uniformly at random from $(\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, $\mathcal{C}$ first generates a polynomial vector $\overrightarrow{b} = (b_1, b_2, \cdots, b_{\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$ s.t. $b_j = 2^u x^{dv}$, for $j = v\tau + u + 1$, where $u \in \{0, \cdots, \tau - 1\}, v \in \{0, \cdots, \gamma - 1\}$.

Let $\varPsi := U(\{-1, 0, 1\})$. After that the challenger picks $\mathbf{y} = (y_1, \cdots, y_\ell) \in \mathbb{Z}_q^\ell$ as in Game 1 and $y_0 = 1$. Then we generate polynomial matrices $\overrightarrow{S_0}, \overrightarrow{S_1}, \cdots, \overrightarrow{S_\ell}$

where $\overrightarrow{S_i} = [\overrightarrow{s_{i,1}}, \overrightarrow{s_{i,2}}, \cdots, \overrightarrow{s_{i,t'}}]^T$ and $\overrightarrow{s_{i,j}} \leftarrow (\varPsi^d[x])^t \times \varPsi^{\gamma\tau}$ for $j = 1, \cdots, \gamma\tau$. Next, set $\overrightarrow{\zeta_i} = \overrightarrow{S_i} \cdot \overrightarrow{a}$ and $\overrightarrow{h_i} = \overrightarrow{S_i} \cdot \overrightarrow{a} + y_i \overrightarrow{b} = \overrightarrow{\zeta_i} + y_i \overrightarrow{b}$.

Then we will show that Game 2 is computationally indistinguishable from Game 1, i.e. $|\Pr[X_2] - \Pr[X_1]| = \mathsf{negl}(n)$.

We are to show that under this construction, $\overrightarrow{h_i} \approx_c U((\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau})$ for $i = 0, 1, \cdots, \ell$. WLOG, we will take the polynomial vector $\overrightarrow{\zeta_i}$ as an example. Let $\overrightarrow{\zeta_i} = (\zeta_1, \zeta_2, \cdots, \zeta_{\gamma\tau})$, where $(\zeta_1, \cdots, \zeta_{\gamma\tau}) \in \mathbb{Z}_q^{n+d-1}[x]$.

For $j = 1, \cdots, \gamma\tau$, $\zeta_j = \overrightarrow{s_{i,j}} \cdot \overrightarrow{a}$. Let $\overrightarrow{s_{i,j}} = [s_1, s_2, \cdots, s_{\gamma\tau}]^T$. So $\zeta_j = \overrightarrow{s_{i,j}} \cdot \overrightarrow{a} = \sum_{k=1}^{t'} s_k \cdot a_k = \sum_{k=1}^{t} s_k \cdot a_k + \sum_{k=t+1}^{t'} s_k \cdot a_k$

By leftover hash lemma (Lemma 2) and our parameter setting ($dt/n = \varOmega(n \log n)$), we can get that $\sum_{k=1}^{t} s_k \cdot a_k \approx_s U(\mathbb{Z}_q[x]^{n+d-1})$. So $(\zeta_1, \cdots, \zeta_{\gamma\tau}) \approx_c U((\mathbb{Z}_q^{n+d-1})^{\gamma\tau})$. Hence, we can get that $\overrightarrow{\zeta_i} \approx_c U((\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau})$ for $i = 0, 1, \cdots, \ell$. Thus, we can get $\overrightarrow{h_i} \approx_c U((\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau})$ for $i = 0, 1, \cdots, \ell$. Therefore, we have $|\Pr[X_2] - \Pr[X_1]| = \mathsf{negl}(n)$.

Game 3. Previously, $\mathcal{C}$ aborts the game at the end of the game if the condition (2) does not hold. In this game, we change the game so that the challenger aborts as soon as the abort condition is satisfied. Since this is only a conceptual change, we have $\Pr[X_3] = \Pr[X_2]$

Game 4. In this game, we change the way the polynomial vector $\overrightarrow{a}$ is chosen. Namely, in Game 4, instead of generating $\overrightarrow{a}$ with trapdoor td using $\mathsf{TrapGen}$, we select $\overrightarrow{a} \overset{\$}{\leftarrow} (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$. By Theorem 1, this only differs negligibly with game 3.

We set $\overrightarrow{S_{\mathsf{id}}} := \overrightarrow{S_0} + \sum_{i=1}^{\ell} d_i \overrightarrow{S_i}$. So it holds that $\overrightarrow{h_{\mathsf{id}}} = \overrightarrow{S_{\mathsf{id}}} \cdot \overrightarrow{a} + F_{\mathbf{y}}(\mathsf{id}) \overrightarrow{b}$. Then we change the way that key extraction queries are answered. If $F_{\mathbf{y}}(\mathsf{id}) = 0$, it aborts as the previous game. Otherwise, it runs $\mathsf{SampleRight}(\overrightarrow{b}, \mathsf{td}, F_{\mathbf{y}}(\mathsf{id}), \overrightarrow{a}, \overrightarrow{S_{\mathsf{id}}}, u, \sigma) \to \overrightarrow{r}$ and returns $\overrightarrow{r}$ to $\mathcal{A}$.

In Game 3, the key $\overrightarrow{r}$ is sampled using $\mathsf{SampleLeft}(\overrightarrow{a}, \overrightarrow{h_{\mathsf{id}}}, \mathsf{td}, u, \sigma) \to \overrightarrow{r}$. By lemma 6, Lemma 7 and the choice of σ, the difference in the output distributions of $\mathsf{SampleRight}$ and $\mathsf{SampleLeft}$ are $\mathsf{negl}(n)$-close. So the above change only differs negligibly in the view of $\mathcal{A}$. Therefore, we have $|\Pr[X_4] - \Pr[X_3]| = \mathsf{negl}(n)$.

Game 5. Game 5 is identical to Game 4 except for the way to generate the challenge ciphertext.

If $F_{\mathbf{y}}(\mathsf{id}^*) = 0$ (i.e. if it does not abort), to create the challenge ciphertext, we first choose $p \overset{\$}{\leftarrow} \mathbb{Z}_q^{<n+2d+k-1}[x]$, $\chi \leftarrow \lfloor D_{\alpha \cdot q} \rceil$, $\chi_1 \leftarrow \lfloor D_{\alpha_1 \cdot q} \rceil$, and noise $e_0, e_1, e_2, \cdots, e_{t''}$, where $e_i \leftarrow \chi^{k+1}$ for $i = 0$, $e_i \leftarrow \chi^{2d+k}[x]$ for $1 \le i \le t$, $e_i \leftarrow \chi^{d+k+1}[x]$ for $t+1 \le i \le t'$, $e_i \leftarrow \chi_1^{d+k+1}[x]$ for $t'+1 \le i \le t''$.

Next, for $1 \le i \le t$ set $c_i = a_i \odot_{2d+k} p + 2e_i$. For $t+1 \le i \le t'$ set $c_i = a_i \odot_{d+k+1} p + 2e_i$. For $t = 0$, set $c_0 = m_+ u \odot_{k+1} p + 2e_i$. For $t'+1 \le i \le t''$, WLOG, we will take c_ι, where $t'+1 \le \iota \le t''$, as an example.

Since $c_\iota = h_{\mathsf{id},\iota} \odot_{d+k+1} p + 2e_\iota$, and $h_{\mathsf{id}} = \overrightarrow{S_{\mathsf{id}}} \cdot \overrightarrow{a}$ as $F_{\mathbf{y}}(\mathsf{id}^*) = 0$, we have $c_\iota = h_{\mathsf{id},\iota} \odot_{d+k+1} p + 2e_\iota = (\sum_{i=1}^{t'} s_i \cdot a_i) \odot_{d+k+1} p + 2e_\iota = \sum_{i=1}^{t} (s_i \cdot a_i) \odot_{d+k+1} p +$

$\sum_{i=t+1}^{t'}(s_i{\cdot}a_i)\odot_{d+k+1}p{+}2e_\iota = \sum_{i=1}^{t} s_i\odot_{d+k+1}(a_i\odot_{2d+k}p){+}\sum_{i=t+1}^{t'} s_i(a_i\odot_{d+k+1} p) + 2e_\iota$, where $(s_1, s_2, \cdots, s_{t'})$ are the entries on the ι-th row of the polynomial matrix $\overrightarrow{S_{\mathsf{id}}}$. The fourth equality holds by Lemma 4, and $s_i \in \mathbb{Z}_q$ for $t+1 \le i \le t'$.

Next, we set $\phi_i = a_i \odot_{2d+k} p$, for $1 \le i \le t$, $\phi_i = a_i \odot_{d+k+1} p$, for $t+1 \le i \le t'$, and use ϕ_i to represent its coefficient vector. Use $\mathbf{e}_\iota$ to represent the coefficient vector of e_ι. Then we set $\overline{\mathbf{c}_\iota} = \sum_{i=1}^{t}(\mathsf{T}^{d,d+k+1}(s_i))^T\overline{\phi_i} + \sum_{i=t+1}^{t'} s_i\overline{\phi_i} + 2\overline{\mathbf{e}_\iota}$. By Lemma 5, we can see that $\mathbf{c}_\iota = \overline{\overline{\mathbf{c}}_\iota}$ is the coefficient vector of c_ι. We can see that Game 4 and Game 5 are identical. So we have $\Pr[X_5] = \Pr[X_4]$.

Game 6. Game 6 is identical to Game 5 except for the way to generate the challenge ciphertext.

If $F_{\mathbf{y}}(\mathsf{id}^*) = 0$ (i.e. if it does not abort), to create the challenge ciphertext, we first choose $p \xleftarrow{\$} \mathbb{Z}_q^{<n+2d+k-1}[x]$, $\chi \leftarrow \lfloor D_{\alpha\cdot q}\rceil$, $\chi_1 \leftarrow \lfloor D_{\alpha_1\cdot q}\rceil$, and noise $e_0, e_1, e_2, \cdots, e_{t''}$, where $e_i \leftarrow \chi^{k+1}$ for $i = 0$, $e_i \leftarrow \chi^{2d+k}[x]$ for $1 \le i \le t$, $e_i \leftarrow \chi^{d+k+1}[x]$ for $t+1 \le i \le t'$.

Next, for $1 \le i \le t$ set $c_i = a_i \odot_{2d+k} p + 2e_i$. For $t+1 \le i \le t'$ set $c_i = a_i \odot_{d+k+1} p + 2e_i$. For $t = 0$, set $c_0 = m_{\mathsf{coin}} + u \odot_{k+1} p + 2e_i$. For $t'+1 \le i \le t''$, WLOG, we will take c_ι, where $t'+1 \le \iota \le t''$, as an example.

First, since $c_i = a_i \odot_{2d+k} p + 2e_i$ for $1 \le i \le t$ and $c_i = a_i \odot_{d+k+1} p + 2e_i$, for $t+1 \le i \le t'$. Use $\mathbf{c}_i$ to represent the coefficient vectors.

Then, we set $\overline{\mathbf{c}_\iota} = \sum_{i=1}^{t} \mathsf{ReRand}((\mathsf{T}^{d,d+k+1}(s_i))^T, \overline{\mathbf{c}_i}, \alpha q, \frac{\alpha_1}{2\sqrt{t'}\alpha}) + \sum_{i=t+1}^{t'} \mathsf{ReRand}(s_i I_{d+k+1}, \overline{\mathbf{c}_i}, \alpha q, \frac{\alpha_1}{2\sqrt{t'}\alpha})$. Set c_ι to be the polynomial whose coefficient vector is $\mathbf{c}_\iota = \overline{\overline{\mathbf{c}}_\iota}$. Since $\alpha_1 = 2\sqrt{t'}(2d+k)\ell\alpha$, $\frac{\alpha_1}{2\sqrt{t'}\alpha} > \ell\sqrt{(2d+k)d} = \sigma_1(\mathsf{T}^{d,d+k+1}(s_i))$ (the inequality holds by Lemma 1), from Lemma 3, we can see that c_ι in this game is distributed statistically close to c_ι in Game 5. So we have $|\Pr[X_6] - \Pr[X_5]| = \mathsf{negl}(n)$.

Game 7. Game 7 is identical to Game 6 except that we change the way the ciphertexts are generated. If $F_{\mathbf{y}}(\mathsf{id}^*) = 0$ (i.e. if it does not abort), to create the challenge ciphertext, we set $c_1, c_2, \cdots, c_t \xleftarrow{\$} \mathbb{Z}_q^{2d+k}$, and $c_{t+1}, \cdots, c_{t'} \xleftarrow{\$} \mathbb{Z}_q^{d+k+1}$ and $t \xleftarrow{\$} \mathbb{Z}_q^{k+1}$, $c_0 = t + m_{\mathsf{coin}}$. Then we generate the rest of the ciphertexts as in Game 6. We can see that $|\Pr[X_7] - \frac{1}{2}| = \mathsf{negl}(n)$ as it contains no information about the coin. Now it remains to show that $|\Pr[X_7] - \Pr[X_6]| = \mathsf{negl}(n)$.

As we will show in Lemma 11, assuming $\text{dMPLWE}_{q,n+2d+k,\mathbf{d},D_{\alpha\cdot q}}$ is hard with degree vector $\mathbf{d} = (d_i)_{i=1}^{t'+1}$ where

$$d_i = \begin{cases} 2d+k, \text{for } 1 \le i \le t \\ d+k+1, \text{for } t+1 \le i \le t' \\ k+1, \text{for } i = t''+1 \end{cases},$$

i.e. $|\Pr[X_7] - \Pr[X_6]| = \mathsf{negl}(n)$.

Analysis. From the above, we can get

$$|\Pr[X_7] - \frac{1}{2}| = |\Pr[X_0] + \sum_{i=0}^{6}(\Pr[X_{i+1}] - \Pr[X_i]) - \frac{1}{2}|$$

$$\geq |\Pr[X_0] - \frac{1}{2}| - \sum_{i=0}^{6}|\Pr[X_{i+1}] - \Pr[X_i]| \geq |\Pr[X_0] - \frac{1}{2}| - \mathsf{negl}(n)$$

Since $|\Pr[X_7] - \frac{1}{2}| = \mathsf{negl}(n)$, we have $|\Pr[X_0] - \frac{1}{2}| = \mathsf{negl}(n)$.

Lemma 11. *For any PPT adversary $\mathcal{A}$, there exists another PPT adversary $\mathcal{B}$ such that $|\Pr[X_7] - \Pr[X_6]| \leq Adv_{\mathcal{B}}^{\mathrm{dMPLWE}_{q,n+2d+k,\mathbf{d},D_{\alpha\cdot q}}}$. In particular, under the $\mathrm{dMPLWE}_{q,n+2d+k,\mathbf{d},D_{\alpha\cdot q}}$ assumption, we have $|\Pr[X_7] - \Pr[X_6]| = \mathsf{negl}(n)$.*

Proof. Suppose $\mathcal{A}$ has a non-negligible advantage in distinguishing Game 6 and 7. We use $\mathcal{A}$ to construct an MPLWE algorithm, denoted $\mathcal{B}$. Recall from Definition 7 that a dMPLWE problem instance is provided as a sampling oracle $\mathcal{O}$ that can be either truly random $\mathcal{O}_\$$ or noisy pseudo-random $\mathcal{O}_s$ for some secret $\mathbb{Z}_q^{<n+2d+k-1}[x]$. The simulator $\mathcal{B}$ uses the adversary $\mathcal{A}$ to distinguish between the two, and proceeds as follows:

Instance. $\mathcal{B}$ requests from $\mathcal{O}$ and receives fresh pairs $(a_1^*, w_1), \cdots, (a_{t'}^*, w_{t'}), (u^*, v^*)$, where $a_i^* \in \mathbb{Z}_q^{<n}$, $w_i \in \mathbb{Z}_q^{<2d+k}$ for $1 \leq i \leq t$, $a_i^* \in \mathbb{Z}_q^{<d+k+1}$, $w_i^* \in \mathbb{Z}_q^{<d+k+1}$ for $t+1 \leq i \leq t'$, and $u^* \in \mathbb{Z}_q^{n+2d-2}$, $v^* \in \mathbb{Z}_q^{<k+1}$.

Setup. Construct master public key mpk as follows:

- Set $a_i = a_i^*$ for $i = 1, \cdots, t'$.
- $\mathcal{B}$ picks $\mathbf{y}$ as in Game 1.
- Generate polynomial vectors $\overrightarrow{h_0}, \cdots, \overrightarrow{h_\ell}$ by picking polynomial matrices $\overrightarrow{S_0}$, $\overrightarrow{S_1}, \cdots, \overrightarrow{S_\ell}$ and generating polynomial vector $\overrightarrow{b}$ as in Game 4.
- Set $u = u^*$.

$\mathcal{B}$ returns mpk to $\mathcal{A}$. $\mathcal{B}$ picks random bit $\mathsf{coin} \leftarrow \{0, 1\}$ and keeps it secret.

Queries. When $\mathcal{A}$ makes a key extraction for id, $\mathcal{B}$ first calculates $F_{\mathbf{y}}(\mathsf{id})$. It aborts and sets $\mathsf{coin}' \xleftarrow{\$} \{0, 1\}$ if $F_{\mathbf{y}}(\mathsf{id}) = 0$. Otherwise, $\mathcal{B}$ generates the private key as in Game 6.

Challenge. When $\mathcal{A}$ makes an encryption query for the challenge identity id^*, $\mathcal{B}$ first computes $F_{\mathbf{y}}(\mathsf{id})$. It aborts and sets $\mathsf{coin}' \xleftarrow{\$} \{0, 1\}$ if $F_{\mathbf{y}}(\mathsf{id}) \neq 0$. Otherwise, it proceeds as follows:

First, it computes $\overrightarrow{S}_{\mathsf{id}^*} = [\overrightarrow{s_1}, \overrightarrow{s_2}, \cdots, \overrightarrow{s_{t'}}]^T$ where $\overrightarrow{s_j} \leftarrow (\chi^d[x])^t \times 0^{\gamma\tau}$ for $j = 1, \cdots, t$, $\overrightarrow{s_j} \leftarrow (\chi^d[x])^t \times \chi^{\gamma\tau}$ for $j = t+1, \cdots, t'$. And we can write $\overrightarrow{s_j}$ as $(s_{j,1}, \cdots, s_{j,t'})$.

For $1 \leq i \leq t'$, set $c_i = w_i$ and use $\mathbf{c}_i$ to represent the coefficient vectors. Set $c_0 = v^* + m_{\mathsf{coin}}$. For $t'+1 \leq i \leq t''$, WLOG, we will take c_ι, where $t'+1 \leq \iota \leq t''$, as an example.

Then, we set $\overline{\mathbf{c}_\iota} = \sum_{i=1}^{t} \mathsf{ReRand}((\mathsf{T}^{d,d+k+1}(s_{\iota,i}))^T, \overline{\mathbf{c}_i}, \alpha q, \frac{\alpha_1}{2\sqrt{t'}\alpha}) + \sum_{i=t+1}^{t'} \mathsf{ReRand}(s_{\iota,i} I_{d+k+1}, \overline{\mathbf{c}_i}, \alpha q, \frac{\alpha_1}{2\sqrt{t'}\alpha})$. And set c_ι to be the polynomial whose coefficient vector is $\mathbf{c}_\iota$. Finally, it returns $(c_0, (c_i)_{i=1}^{t''})$ to $\mathcal{A}$.

Guess. At last, $\mathcal{A}$ outputs its guess $\widetilde{\mathsf{coin}}$ if the abort condition has not been satisfied. Then $\mathcal{B}$ sets $\mathsf{coin}' = \widetilde{\mathsf{coin}}$. Finally, $\mathcal{B}$ outputs 1 if $\mathsf{coin} = \mathsf{coin}'$ and 0 otherwise.

Analysis. We can see that $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in Game 6 if $\mathcal{O} = \mathcal{O}_s$ and Game 7 if $\mathcal{O} = \mathcal{O}_\$$. Note that both games only differ in the generation of the challenge ciphertext. Furthermore, we can see that the generation of mpk, the queries, and the challenge phase are the same in both cases. So if $\mathcal{B}$ is able to distinguish between Game 6 and Game 7 with non-negligible probability, i.e. $|\Pr[X_6] - \Pr[X_7]| > \mathsf{negl}(n)$, $\mathcal{B}$ can solve $\text{dMPLWE}_{q,n+2d+k,\mathbf{d}}$ problem with non-negligible probability. And this contradicts to Theorem 5. Therefore, $|\Pr[X_6] - \Pr[X_7]| = Adv_{\mathcal{B}}^{\text{dMPLWE}_{q,n+2d+k,\mathbf{d},D_{\alpha \cdot q}}} = \mathsf{negl}(n)$ as desired. □

Remark 1. We use fixed constant ℓ to denote the length of the identity. Picking $d, k = \Theta(n)$, $t = \log(n)$, and $\alpha, \alpha_1, \sigma, q$ following the parameter settings. By construction, the IBE scheme we proposed has master public key mpk of size $O(n \log(n))$, secret key for the identity $\mathsf{sk}_{\mathsf{id}}$ and ciphertext c of size $n \log(n)$.

5 Conclusion

In this paper, we presented the first adaptively secure IBE scheme based on MPLWE, filling a gap in the literature as only selectively-secure IBEs from MPLWE has been proposed previously. Our scheme inherent the typical advantages of MPLWE, namely, it has comparable efficiency to its PLWE counterparts while offering stronger security guarantees. However, the master public key of our scheme is larger than the state-of-the-art PLWE-based constructions. As such, one direction for future research is to develop adaptively secure IBE schemes with compact master keys based on MPLWE.

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_28
2. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: Miller, G.L. (ed.), Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, 22–24 May 1996, pp. 99–108. ACM (1996)
3. Bai, S., et al.: MPSign: a signature from small-secret middle-product learning with errors. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 66–93. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_3

4. Bellare, M., Ristenpart, T.: Simulation without the artificial abort: simplified proof and improved concrete security for waters' IBE scheme. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 407–424. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_24
5. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_14
6. Boneh, D., Boyen, X.: Secure identity based encryption without random oracles. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 443–459. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_27
7. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_13
8. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 255–271. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_16
9. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 523–552. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_27
10. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 360–363. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45325-3_32
11. Das, D., Au, M.H., Zhang, Z.: Ring signatures based on middle-product learning with errors problems. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2019. LNCS, vol. 11627, pp. 139–156. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_8
12. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.), Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, 17–20 May 2008, pp. 197–206. ACM (2008)
13. Hofheinz, D., Kiltz, E.: Programmable hash functions and their applications. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 21–38. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_2
14. Katsumata, S., Yamada, S.: Partitioning via non-linear polynomial functions: more compact IBEs from ideal lattices and bilinear maps. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 682–712. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_23
15. Le, H.Q., Duong, D.H., Susilo, W., Pieprzyk, J.: Trapdoor delegation and HIBE from middle-product LWE in standard model. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 2020, Part I. LNCS, vol. 12146, pp. 130–149. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57808-4_7
16. Lombardi, A., Vaikuntanathan, V., Vuong, T.D.: Lattice trapdoors and IBE from middle-product LWE. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 24–54. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_2
17. Lyubashevsky, V.: Digital signatures based on the hardness of ideal lattice problems in all rings. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 196–214. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_7

18. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
19. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
20. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on gaussian measures. In: 45th Symposium on Foundations of Computer Science (FOCS 2004), 17–19 October 2004, Rome, Italy, Proceedings, pp. 372–381. IEEE Computer Society (2004)
21. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 34:1–34:40 (2009)
22. Roşca, M., Sakzad, A., Stehlé, D., Steinfeld, R.: Middle-product learning with errors. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 283–297. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_10
23. Rosca, M., Stehlé, D., Wallet, A.: On the ring-LWE and polynomial-LWE problems. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 146–173. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_6
24. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_5
25. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 617–635. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_36
26. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_7
27. Waters, B.: Dual system encryption: realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_36
28. Yamada, S.: Adaptively secure identity-based encryption from lattices with asymptotically shorter public parameters. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 32–62. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_2
29. Yamada, S.: Asymptotically compact adaptively secure lattice IBEs and verifiable random functions via generalized partitioning techniques. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 161–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_6
30. Zhang, J., Chen, Yu., Zhang, Z.: Programmable hash functions from lattices: short signatures and IBEs with small key sizes. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 303–332. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_11

Post-Quantum Cryptography

Quantum-Access Security of Hash-Based Signature Schemes

Quan Yuan[1](✉), Mehdi Tibouchi[2,3], and Masayuki Abe[2,3]

[1] The University of Tokyo, Tokyo, Japan
yuanquan@g.ecc.u-tokyo.ac.jp
[2] NTT Social Informatics Laboratories, Tokyo, Japan
[3] Kyoto University, Kyoto, Japan

Abstract. In post-quantum cryptography, hash-based signature schemes are attractive choices because of the weak assumptions. Most existing hash-based signature schemes are proven secure against post-quantum chosen message attacks (CMAs), where the adversaries are able to execute quantum computations and classically query to the signing oracle. In some cases, the signing oracle is also considered quantum-accessible, meaning that the adversaries are able to send queries with superpositions to the signing oracle. Considering this, Boneh and Zhandry propose a stronger security notion called existential unforgeability under quantum chosen message attacks (EUF-qCMA). We call it quantum-access security (or Q2 security in some literature). The quantum-access security of practical signature schemes is lacking in research, especially of the hash-based ones. In this paper, we analyze the quantum-access security of hash-based signature schemes in two directions. First, we show concrete quantum chosen message attacks (or superposition attacks) on existing hash-based signature schemes, such as SPHINCS and SPHINCS+. The complexity of the attacks is obviously lower than that of optimal classical chosen message attacks, implying that quantum chosen message attacks are more threatening than classical ones to these schemes. Second, we propose a simple variant of SPHINCS+ and give security proof against quantum chosen message attacks. As far as we know, it is the first practical hash-based stateless signature scheme against quantum chosen message attacks with concrete provable security.

Keywords: hash-based signatures · quantum security · post-quantum cryptography · digital signatures · superposition attacks

1 Introduction

1.1 Background

Quantum-Access Security of Signature Schemes. Signature schemes [20] are essential primitives in cryptography. In the security analysis of a signature

This work is carried out during his PhD study in Kyoto University, Japan.

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 343–380, 2023.
https://doi.org/10.1007/978-3-031-35486-1_16

scheme, one usually considers existential unforgeability under chosen message attacks (EUF-CMA). In this model, the adversary can query messages to a signing oracle. After the signing queries, the adversary is required to output a valid signature σ^* for a fresh message m^*. We say a scheme is EUF-CMA if any polynomial-time adversary succeeds with negligible probability.

Quantum attacks on cryptographic schemes are usually classified into two types [12,23,24]. In the first type, the adversary can use quantum computers to execute some offline algorithms or evaluate some functions and send classical queries to the online oracles. This is called post-quantum security (or Q1 security in some literature). In the second type, the queries to the oracles are also in superpositions. This is called quantum-access security (or Q2 security). In signature schemes, the above EUF-CMA security is a kind of Q1 security. In recent years, there is a large amount of research on Q2 security of various cryptographic primitives, including pseudorandom functions [35], message authentication codes [10], encryption schemes [11], signature schemes [2,11,15,17] and so on.

In 2013, Boneh and Zhandry [11] propose a Q2 security notion called existential unforgeability under *quantum* chosen message attacks (EUF-qCMA). In this experiment, the adversary is required to output $(q_s + 1)$ forgeries for distinct messages after q_s quantum signing queries. They also show a separating example, implying that EUF-qCMA is a strictly stronger security notion. However, to the best of our knowledge, there is no concrete evidence to show that a Q2 attacker can be obviously stronger than a Q1 one for *practical* signature schemes (although it is intuitively true).

Hash-Based Signatures. Hash-based signature (HBS) schemes are ones whose security is solely based on secure hash functions rather than mathematical hard problems. Because of the weak assumptions and resistance to quantum attacks, HBS schemes are fairly competitive in post-quantum cryptography.

The first practical stateless HBS scheme is SPHINCS [7]. It is based on HORST (Hash to Obtain Random Subsets with Trees [31]), a few-time stateless HBS scheme, where the number of signing operations is limited by a small constant. A key pair of SPHINCS can issue at most 2^{50} signatures and provide 128-bit quantum security[1]. So far, the state-of-art stateless HBS scheme is SPHINCS+ [3,8], a variant using improved building blocks such as few-time signatures and tweakable hash functions. It behaves better in terms of the signature size, efficiency, security proof and the maximum number of signing executions (reaching to 2^{64}). Recently, SPHINCS+ is selected as one of the NIST post-quantum cryptography standardization.

In terms of the quantum-access security of HBS schemes, Boneh and Zhandry [11] prove the EUF-qCMA security of Lamport's scheme [28] and MSS [30]. The

[1] In this paper, we always focus on signature schemes with security level 5 in NIST post-quantum cryptography standardization. That is, breaking the security is at least as hard as finding a preimage of AES-256. It implies 128-bit security, which means that breaking the security requires about 2^{128} hash queries. In particular, we focus on SPHINCS-256 and SPHINCS+-256s/256f.

former is a one-time HBS scheme, and the latter is a stateful one. Hopefully, other stateful HBS schemes (such as XMSS [14]) are also EUF-qCMA since they are essentially variants of MSS and the structures are similar. However, when it comes to stateless schemes, the cases become different. (The authors proved the EUF-qCMA security of stateless MSS, but stateless MSS is far from efficient in practice.) For practical stateless HBS schemes, such as SPHINCS and SPHINCS+, the quantum-access security still lacks research.

Note that Boneh and Zhandry [11] proposed a generic construction of EUF-qCMA schemes from UUF-RMA (universally unforgeable under random message attacks) schemes by introducing a hash function modeled as a quantum random oracle. However, the construction causes large security loss, which will be especially expensive for HBS schemes, whose security are highly related to the number of issued signatures. For instance, suppose we want an EUF-qCMA scheme such that the probability of breaking the security is under 2^{-10} when $q_s \leq 2^{10}$ and $q_H \leq 2^{30}$ (number of signing queries and hash computations respectively), which is a fairly weak requirement in practice. If we use the above generic construction, we need a 2^{117}-time UU-RMA scheme with 254-bit quantum security, which far exceeds the security of SPHINCS+.

1.2 Our Contributions

In this paper, we give positive and negative results on quantum-access security of HBS schemes.

- First, we show some qCMAs on two HBS schemes, SPHINCS and SPHINCS+. The complexity of our attacks is much lower than the security in the classical setting. We thus conclude that quantum chosen message attacks are more threatening to security than classical CMAs for the two schemes.
- Second, we propose SPHINCS-FORS, a simple variant of SPHINCS+, whose quantum-access security can be proven. We give formal security proof and concrete security in sense of EUF-CMA and EUF-qCMA. As far as we know, it is the first practical stateless HBS scheme with provable security against qCMAs.

1.3 Main Techniques

Quantum Chosen Message Attacks on HBS Schemes. We first give an outline of SPHINCS and SPHINCS+. Roughly speaking, SPHINCS(+) introduces a stateful signature HT and implicit 2^h instances of few-time signature key pairs (HORST in SPHINCS and FORS in SPHINCS+). HT is for signing the public keys of the few-time signatures, and the few-time signatures are for signing the messages. In each signing operation, it (pseudo-)randomly picks an index $idx \in \{0,1\}^h$ and uses the few-time signature key pairs labeled idx to sign a message. The essential idea is that the index is picked randomly, and thus, each few-time signature key pair is not used too many times in the signing operations.

The core idea of our attack is to obtain enough few-time signatures associated with a single index. In SPHINCS, the index is calculated by pseudorandom functions on the message m and included as a part of the SPHINCS signature. Although the pseudorandom functions cannot be directly evaluated by adversaries without the secret key, they can be evaluated by signing queries. Fix some index $idx^* \in \{0,1\}^h$. A quantum-access adversary can search a message m^* mapping to idx^* by Grover's algorithm with $O(2^{h/2})$ quantum queries to the signing oracle.

Thus, our quantum chosen message attack on SPHINCS runs as follows. First, try to obtain enough number (say r) of messages m_i^* mapping to an index idx^* by querying the signing oracle. This requires $O(r2^{h/2})$ quantum signing queries. Then, send m_i^* (with pure states) to the signing oracle separately, and obtain r HORST signatures associated with idx^*. This requires r signing queries. When r is large enough, the HORST secret key associated with idx^* will be completely revealed, and the adversary can forge a signature for*any* message. Finally, generate enough SPHINCS signatures associated with index idx^* to meet the requirement of one-more forgeries. Note that optimal classical chosen message attacks on SPHINCS require approximately 2^{128} hash queries (when at most 2^{50} signatures are issued). The query complexity of our attacks are much lower than that of classical attacks (see Table 1, Our Attacks (PO)).

Result 1. *For some integer r, there exists a quantum chosen message attack on SPHINCS such that*

$$q_s = O(r2^{h/2}), \quad q_H = O((1-e^{-\frac{kr}{t}})^{-\frac{k}{2}} r2^{\frac{h}{2}}),$$

where q_s and q_H denotes the number of quantum signing queries and quantum hash queries, respectively.

The attack on SPHINCS+ is similar. The difference is that the index of a SPHINCS+ signature is not directly included. Instead, it is calculated by $h_0(z||m)$, where h_0 is a hash function mapping to $\{0,1\}^h$ and z is a (pseudo-) randomizer determined by the message m.[2] Since z is included in the signature, the index can be evaluated by a signing query (calculating z) and an additional h_0 query. Then, use Grover's algorithm to find enough FORS signatures associated with an index idx^*. Finally, to generate a forgery (m_i^*, σ_i^*), the adversary needs to find a corresponding randomizer z_i^* such that $h_0(z_i^*||m_i^*) = idx^*$. It requires $O(2^{h/2})$ quantum queries to h_0 by using Grover's algorithm for each forgery. Thus, the attack on SPHINCS+ requires more hash queries than that on SPHINCS (see Table 1, Our Attacks (PO)).

Result 2. *For some integer r, there exists a quantum chosen message attack on SPHINCS+ such that*

$$q_s = O(r2^{h/2}), \quad q_H = O((1-e^{-\frac{r}{t}})^{-\frac{k}{2}} r2^{h}).$$

[2] SPHINCS+ has a probabilistic version where z is not determined by the message. Our attack only works on the deterministic version of SPHINCS(+).

In addition, Alagic et al. [2] propose another security notion called blind unforgeability (BU) considering quantum chosen message attacks.[3]. Informally, the (quantum-accessible) signing oracle now returns $\perp$ for messages in a ε-fraction blind subset of the message space, and it is infeasible to forge a signature for a message in this blind subset. To distinguish the two, we call the previous security model against qCMAs as PO model (Plus-one model). In our study, we also give similar attacks on SPHINCS and SPHINCS+ in the BU model, and the time complexity of the attacks is much lower than that in the PO model (see Table 1, Our Attacks(BU)).

By implementing the parameters in SPHINCS [7] and SPHINCS+ v.3 [3], we give comparisons among the attacks in the EUF-CMA model, the PO model, and the BU model. See more details in Table 1.

Table 1. Comparisons between our qCMAs and the CMA security of SPHINCS(+), which implies the bounds of tolerable hash queries. q_s and q_H denote the number of (quantum) signing queries and quantum hash queries, respectively. For example, if the adversary issues 2^{43} quantum signing queries to the signing oracle and more than 2^{43} quantum hash queries, then SPHINCS-256 will be broken in the PO model. "Small" means that the attack only requires a polynomial number of queries.

Scheme	CMA Security		Our Attack (PO)		Our Attack (BU)	
	$\log q_s$	$\log q_H$	$\log q_s$	$\log q_H$	$\log q_s$	$\log q_H$
SPHINCS-256 [7]	50	128	43	43	43	Small
SPHINCS+-256s [3]	64	128	48	80	43	43
SPHINCS+-256f [3]	64	128	46	80	42	42

A Provably Secure Stateless HBS Scheme Against Quantum Chosen Message Attacks. Although we show qCMAs on SPHINCS and SPHINCS+, the time complexity of the attacks does not imply the precise security levels of the schemes against qCMAs. Indeed, it only implies *upper bounds* of the security. We hope to construct a *provably* secure scheme, whose generic security against qCMAs can be guaranteed and *lower-bounded*.

We start with the security analysis of few-time signature schemes, such as HORS [31] and FORS [8]. Most the practical few-time signature schemes are related to (variants of) subset resilient hash functions (SRH) [31,33]. Unfortunately, their quantum-access security cannot be reduced to subset resilience directly. To solve this problem, we propose a variant of subset resilience called *weak subset resilience* (wSR). We find that the EUF-qCMA security of a randomized version of FORS (say rFORS, implicitly contained in SPHINCS+) can

[3] The notion is proposed for message authentication codes but can also be extended to signature schemes.

then be reduced to wSR and some additional common security notions for hash functions (such as multi-target collision resistance, mTCR).

Since wSR is a new notion and lack of research, the generic security needs to be evaluated. We solve this problem by the quantum query lemma [32]. As a result, the generic security of rFORS against r-time qCMAs can be bounded by

$$\mathrm{Adv}_{\mathsf{rFORS},r,q}^{\mathsf{EUF\text{-}qCMA}}(\mathcal{A}) \leq O\left(q^{2(r+1)}\left(\frac{r^2}{t}\right)^k + \frac{q^2kt^r}{2^n}\right), \tag{1}$$

where q denotes the number of queries to the hash functions (modeled as random oracles) and k, t, n are parameters. The first term of Eq. (1) comes from the generic security of wSR. The second one comes from mTCR and the reduction. They are negligible when r is a small constant number. (In practice, $t \approx 2^{\sqrt{n}}$ and $k \approx \sqrt{n}$.)

(To the best of our knowledge, there is no quantum generic security bound of subset resilience. We also fill this gap, which could be independently interesting. For example, it immediately implies concrete security of the related few-time HBS schemes in terms of post-quantum EUF-CMA.)

We then turn to many-time schemes. The main reason for the insecurity against the above qCMAs is that the index is determined by the message due to the pseudorandom functions. A natural idea to resist these attacks is to introduce additional randomness into the process of choosing the index. Here, we use a simpler approach: we directly replace the index with randomness that is *independent* of the message and include it in the signature. Now, the signatures in a quantum signing response share a common few-time signature key pair in a signing query. The EUF-qCMA security is thus reduced to the EUF-qCMA security of the related few-time signature scheme, rFORS, which can be bounded by Eq. (1).

Although the new variant can avoid some qCMAs, it brings other risks. Since the index is directly included in the signature, an adversary can arbitrarily choose an index in the forgery. Thus, the EUF-CMA security of the new variant needs to be re-analyzed.

Similarly, the EUF-CMA security of our new variant can be reduced to the EUF-CMA security of rFORS. The remaining work is evaluating the EUF-CMA security of rFORS. Indeed, the EUF-CMA security of rFORS can be tightly reduced to a variant of subset resilience called *extended Target Subset Resilience* (eTSR). Unfortunately, the generic security of eTSR cannot be (tightly) evaluated by the same approach. We use the adaptive reprogramming lemma [21] to solve this problem. The EUF-CMA security of rFORS is then bounded by

$$\mathrm{Adv}_{\mathsf{rFORS},r,q}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) \leq O\left(\sqrt{\frac{q}{2^n}} + q^2\left(\frac{r}{t}\right)^k + \frac{q^2}{2^n}\right). \tag{2}$$

The first two terms of Eq. (2) comes from the generic security of eTSR, and the third one comes from mTCR.

Equipped with rFORS, the message-independent random index, and the same HT as in SPHINCS+, we get our new variant called SPHINCS-FORS. (The name is given since the framework looks like SPHINCS (not SPHINCS+) equipped with rFORS.) The outline of the security analysis of SPHINCS-FORS is summarized in Fig. 1.

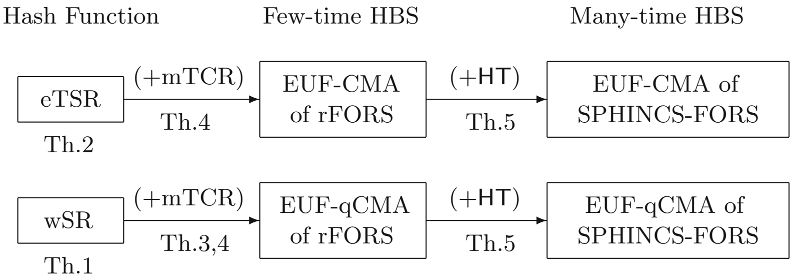


Fig. 1. The security proof sketch of SPHINCS-FORS.

Finally, the generic security of SPHINCS-FORS is evaluated as follows.

Result 3. *Let the hash functions in SPHINCS-FORS be modeled as quantum random oracles. For any adversary $\mathcal{A}$, it holds that*

$$Adv^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SPHINCS\text{-}FORS},q_s,q_H}(\mathcal{A}) \leq O\left(q_s\sqrt{\frac{q_H+q_s}{2^n}} + \frac{q_H}{2^{n/2}} + q_H^2 \sum_{r=0}^{q_s} p(r,q_s)\left(\frac{r}{t}\right)^k\right),$$

$$Adv^{\mathsf{EUF\text{-}qCMA}}_{\mathsf{SPHINCS\text{-}FORS},q_s,q_H}(\mathcal{A}) \leq O\left(\frac{q_H}{2^{n/2}} + \sum_{r=0}^{q_s} p(r,q_s)\cdot \min\left\{q_H^{2(r+1)}\left(\frac{r^2}{t}\right)^k + \frac{q_H^2 k t^r}{2^n}, 1\right\}\right),$$

where $p(r,q_s) = \min\{2^{r(\log q_s - h)+h-\log r!}, 2^h\}$.

For EUF-CMA security, we expect it to reach 128-bit security as SPHINCS+, which means that for fixed q_s (e.g., $q_s = 2^{64}$), $\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SPHINCS\text{-}FORS},q_s,q_H}(\mathcal{A})$ reaches to a constant only if q_H reaches 2^{128}. When $n = 256$, the dominant term is $q_H^2 \sum_{r=0}^{q_s} p(r;q_s)(\frac{r}{t})^k$. We adapt the parameters such that $\sum_{r=0}^{q_s} p(r,q_s)(\frac{r}{t})^k = 2^{-256}$, and then the resulting scheme provides 128-bit security against CMA.

It is more complicated for EUF-qCMA security. The dominant term is also the one with the summation notion, which is larger than the above one. When r is small, $\min\{q_H^{2(r+1)}(\frac{r^2}{t})^k + \frac{q_H^2 k t^r}{2^n}, 1\}$ is negligibly small. When r becomes large, $p(r,q_s)$ becomes negligibly small and the righthand term is always bounded by 1, so the product is still small. Thus, the sum of the products remains bounded if the parameters are well stated.

We give instantiation of SPHINCS-FORS by adapting the parameters in Table 2. We observe that SPHINCS-FORS has larger signature size and running time, but can provide provable security in sense of EUF-qCMA.

Table 2. Comparison between (deterministic) SPHINCS+ and our variants against quantum chosen message attacks. $\log q_H \leq a$ means that there exists a quantum chosen message attack with 2^a quantum hash queries. It implies an upper bound of the security level (without security proof). $\log q_H \geq a$ means that any quantum chosen message attack requires *at least* 2^a hash queries. It implies a lower bound of the security level (with security proof). Note that all the schemes in this table can provide at least 128-bit EUF-CMA security when $\log q_s \leq 64$.

Scheme	Parameters				Security		Size	Provably Secure?
	k	$\log t$	h	n	$\log q_s$	$\log q_H$		
SPHINCS+-256s	22	14	64	256	48	≤80	29272	✗
SPHINCS-FORS v1 (Ours)	32	17	128	256	48	≥80	47072	✓
SPHINCS+-256s∗	22	14	104	256	64	≤128	41792	✗
SPHINCS-FORS v2 (Ours)	48	18	128	384	64	≥128	101424	✓

1.4 Related Work

HBS schemes have a long history that begins with Lamport's one-time signature scheme [28]. SPHINCS [7] is the first practical stateless HBS using Goldreich's framework [19]. There are several variants of SPHINCS [5,8,26,38]. The tight security proof is recently given in the post-quantum setting [25].

The generic security of hash functions is the core of analyzing the concrete security of HBS schemes. There is previous work proving the post-quantum generic security [1,21,25,27,32,36,37]. Especially, SPHINCS-family is related to subset resilience [4,33]. In recent concurrent work, Bouaziz-Ermann and Grilo et al. [13] shows quantum generic security of (restricted) subset resilience when the range of the "partial" hash (say t) is strictly exponential. Unfortunately, their proof fails since t is not large enough in a practical HBS scheme. (For instance, their proof once causes a term of reduction loss $O(t^{1/48})$, which is considered a negligible function when t is exponential. However, t is instantiated by 2^{14} in SPHINCS-256s, and thus the above term cannot be ignored.)

Quantum-access security is first considered for pseudorandom functions [35] and then generalized to message authentication codes and signatures [10,11]. After that, other quantum-access security notions are proposed [2,18] for message authentication codes (and they can also be extended to fit signature schemes). In particular, the blind unforgeability of Lamport's scheme, WOTS, and GPV signatures has been evaluated [15,29].

1.5 Organizations

In Sect. 2, we introduce the basic preliminaries and the security notions for signature schemes in the quantum-access settings. In Sect. 3, we introduce and propose the subset-resilient hash function and its variants. In Sect. 4, we introduce hash-based signature schemes. In Sect. 5, we propose quantum-access attacks on SPHINCS(+). In Sect. 6, we give generic quantum security bound for few-time HBS and eventually construct a provably quantum-access-secure variant of SPHINCS+. Due to the space limitation, we omit some of definitions and security proof in this paper. See details in the full version [34].

2 Preliminaries

2.1 Basic Preliminaries

Notations. For a set X, $|X|$ denotes the cardinality of X and $x \leftarrow X$ means that x is uniformly chosen from X. For integer $n \in \mathbb{N}$, denote $[n] = \{1, ..., n\}$. We say that $\epsilon : \mathbb{N} \rightarrow \mathbb{R}^+$ is negligible function if for every constant $c > 0$, there exists $N_c > 0$ such that $\epsilon(n) < n^{-c}$ holds for all $n > N_c$. We denote by $||$ the concatenation operation.

Random Oracle Model. In the (classical) random oracle model [6], a random function H is uniformly chosen at the beginning and one can compute H by querying the random oracle. In quantum random oracle model [9], H is quantum-accessible, i.e., one can query $\sum_{x,y} \psi_{x,y} |x, y\rangle$ to the quantum random oracle, and obtains $\sum_{x,y} \psi_{x,y} |x, y \oplus H(x)\rangle$ in response.

Hash Functions. In this paper, we frequently use hash functions mapping to k (ordered) elements of set $[t]$ (where $t = 2^\tau$ for some integer τ). We can simply sample such a function by sampling a hash function $H' : \{0,1\}^* \rightarrow \{0,1\}^{k \cdot \log t}$, splitting the output into k short strings of length τ, and then mapping the short strings into integers in $[t]$.

In the following, we denote a function $H : \{0,1\}^* \rightarrow [t]^k$ by $H = (h_1, ..., h_k)$, where $h_i : \{0,1\}^* \rightarrow [t]$ denotes the "partial" function mapping to the i-th element of the output of H. To avoid ambiguity, we always use capital letters (such as H) to denote a hash function and their corresponding small letters with a subscript (such as h_i) as the partial functions. For instance, in the case that $\mathcal{H}$ is a hash function family mapping to $[t]^k$, $(h_1, ..., h_k) \leftarrow \mathcal{H}$ means sampling $H \leftarrow \mathcal{H}$ and letting $H = (h_1, ..., h_k)$, rather than sampling k functions from $\mathcal{H}$.

2.2 Security Notions for Hash-Based Signature Schemes

Let $\Gamma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ be a signature scheme. SigO denotes the signing oracle that computes $\mathsf{Sign}(sk, m)$ where sk is the secret key. If SigO is quantum-accessible, say $|\mathsf{SigO}\rangle$, it means that

$$|\mathsf{SigO}\rangle : \sum_{m,t} \psi_{m,t} |m, t\rangle \mapsto \sum_{m,t} \psi_{m,t} |m, t \oplus \mathsf{Sign}(sk, m)\rangle .$$

Especially, if Sign is probabilistic, SigO replies $\sum_{m,t} \psi_{m,t} |m, t \oplus \mathsf{Sign}(sk, m; r)\rangle$ with a random seed r for a query $\sum_{m,t} \psi_{m,t} |m, t\rangle$.

This paper focuses on hash-based signature (HBS) schemes. The security of an HBS scheme is related to the number of hash queries from the adversary. Let q_s and q_H respectively denote the maximum number of signing queries and hash queries. The security is defined as follows.

Experiment $\mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,q_s,q_H}(1^n,\mathcal{A})$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{SigO}}(pk)$
If m^* has not been queried to SigO and $\mathsf{Ver}(pk, m^*, \sigma^*) = 1$, **return** 1, otherwise **return** 0.

Experiment $\mathsf{Exp}^{\mathsf{EUF\text{-}qCMA}}_{\Gamma,q_s,q_H}(1^n,\mathcal{A})$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$
$\{(m_j, \sigma_j)\}_{j\in[r+1]} \leftarrow \mathcal{A}^{|\mathsf{SigO}\rangle}(pk)$
If m_j's are distinct and for $\forall j \in [q_s + 1]$, $\mathsf{Ver}(pk, m_j, \sigma_j) = 1$, **return** 1, otherwise **return** 0.

Definition 1. *([11]) Let Γ be a signature scheme. We say it is existentially unforgeable under chosen message attacks (EUF-CMA) (or quantum chosen message attacks (EUF-qCMA)) if for all probabilistic polynomial-time adversary $\mathcal{A}$, $\Pr[\mathit{Exp}^{\mathit{EUF\text{-}CMA}}_{\Gamma,q_s,q_H}(1^n,\mathcal{A})] \leq \boldsymbol{negl}(n)$ (or $\Pr[\mathit{Exp}^{\mathit{EUF\text{-}qCMA}}_{\Gamma,q_s,q_H}(1^n,\mathcal{A})] \leq \boldsymbol{negl}(n)$) holds, where **negl** is a negligible function.*

In addition, we introduce a security notion called blind unforgeability [2]. Let $\varepsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ be an efficiently computable function. $B_{\varepsilon,n}$ be a subset of the message space $\mathcal{M}_n$ that is selected by placing each $m \in M_n$ independently with probability $\varepsilon(n)$. Define the blind signing oracle $B_{\varepsilon,n}\mathsf{SigO}$ as follows:

$$B_{\varepsilon,n}\mathsf{SigO} : m \mapsto \begin{cases} \mathsf{Sign}(sk, m) & (m \notin B_{\varepsilon,n}), \\ \perp, & (otherwise). \end{cases}$$

Similarly, when $B_{\varepsilon,n}\mathsf{SigO}$ is quantum-accessible, say $|B_{\varepsilon,n}\mathsf{SigO}\rangle$, it maps

$$|B_{\varepsilon,n}\mathsf{SigO}\rangle : \sum_{m,t} \psi_{m,t} |m, t\rangle \mapsto \sum_{m,t} \psi_{m,t} |m, t \oplus B_{\varepsilon,n}\mathsf{SigO}(m)\rangle .$$

Then, the experiment of blind unforgeability under quantum chosen message attacks is as follows:

Experiment $\mathsf{Exp}^{\mathsf{BU\text{-}qCMA}}_{\Gamma,q_s,q_H}(1^n,\mathcal{A})$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{|B_{\varepsilon,n}\mathsf{SigO}\rangle}(pk)$
If $m \in B_{\varepsilon,n} \wedge \mathsf{Ver}(pk, m^*, \sigma^*) = 1$, **return** 1, otherwise **return** 0.

Remark 1. In this paper, we mainly discuss EUF-qCMA security instead of BU except in Sect. 5.3. Apart from this subsection, we use the EUF-qCMA model in Definition 1 to evaluate the security against qCMAs by default.

We omit the security parameter 1^n in the inputs of the algorithms and experiments for simplicity.

2.3 Toolbox

In this section, we show some useful lemmas related to quantum computations and quantum random oracles.

Lemma 1. *(Quantum query complexity lemma. [32]) Let H be a random function mapping $\mathcal{X}$ to $\mathcal{Y}$, r be a positive integer, and R be a relation over $\mathcal{Y}^r$. For any quantum adversary $\mathcal{A}$ which can query H at most q times, it holds that*

$$\Pr_H\left[x_1,...,x_r \text{ are distinct} \wedge (H(x_1),...,H(x_r)) \in R|(x_1,...,x_r) \leftarrow \mathcal{A}^{|H\rangle}\right]$$
$$\leq (2q+1)^{2r}\Pr\left[\exists\pi \in \mathsf{Perm}([r]) \text{ s.t. } (y_{\pi(1)},...,y_{\pi(r)}) \in R|(y_1,...,y_r) \leftarrow \mathcal{Y}^r\right],$$

where $\mathsf{Perm}([r])$ denotes the set of permutations of $[r]$.

Corollary 1. *Let $\mathcal{F}$ be an efficient function esemble modeled by a quantum random oracle. For any quantum adversary $\mathcal{A}$, it holds that*

$$Adv^{\mathsf{OW}}_{\mathcal{F},q}(\mathcal{A}) \leq (2q+1)^2 \cdot 2^{-n}.$$

The next lemma show that if we perform a partial measurement in the process of a quantum algorithm and the measurement obtains one of t outcomes, then the final output of the algorithm will remain unchanged with probability at least $1/t$.

Lemma 2. *[11] Let A be a probabilistic quantum algorithm. Let A' be another algorithm described as follows: A' runs as A but pauses it at an arbitrary stage of execution, performs a partial measurement that obtains one of t outcomes, and then resumes A. For any x, it holds that*

$$\Pr_{A'}[x \leftarrow A'] \geq \Pr_A[x \leftarrow A]/t.$$

Definition 2. *[27] Let $\mathcal{F} \triangleq \{f : \{0,1\}^m \to \{0,1\}\}$ be the collection of all boolean functions with input space $\{0,1\}^m$. Let $\lambda \in [0,1]$ be a constant. Define a family of distributions D_λ on $\mathcal{F}$ such that for $f \leftarrow D_\lambda$, $\forall x \in \{0,1\}^m$, $f(x) = 1$ with probability λ and $f(x) = 0$ with probability $1 - \lambda$.*

Lemma 3. *[27] Let $\mathcal{A}$ be an algorithm $\mathcal{A}$ issuing q quantum queries to $f(\cdot)$. Define*

$$Adv^{\mathsf{Avg\text{-}Search}_\lambda}_{\mathcal{F},q}(\mathcal{A}) \triangleq \Pr_{f\leftarrow D_\lambda}[f(x) = 1|x \leftarrow \mathcal{A}^f].$$

Then, for any adversary $\mathcal{A}$, it holds that

$$Adv^{\mathsf{Avg\text{-}Search}_\lambda}_{\mathcal{F},q}(\mathcal{A}) \leq 8\lambda(q+1)^2.$$

Next we introduce the adaptive reprogramming lemma [21]. It shows that if we reprogram the random oracle in some partially random records, then an adversary is hard to tell the difference. For an oracle $\mathsf{O} : X \to Y$, $x \in X$ and $y \in Y$, denote $\mathsf{O}^{x\to y}$ as an oracle that is the same as O except that it maps x to y. The adaptive reprogramming lemma is as follows.

Lemma 4. *(Adaptive reprogramming lemma. [21]) Let X_1, X_2 and Y be finite sets, $\mathsf{O}_0 : X_1 \times X_2 \rightarrow Y$ be a random oracle and Repro_b be the adaptive reprogramming game depicted in Fig. 2. Let $\mathcal{A}$ be an algorithm issuing q quantum queries to O_b and R classical queries to $\mathsf{Reprogram}$. Then, the probability of distinguishing b is at most*

$$|\Pr[\mathit{Repro}_1^{\mathcal{A}} = 1]| - |\Pr[\mathit{Repro}_0^{\mathcal{A}} = 1]| \leq \frac{3R}{2}\sqrt{\frac{q}{|X_1|}}.$$

```
Game Repro_b                          Reprogram(x_2)

$O_1 := O_0$                          $(x_1, y) \leftarrow X_1 \times Y$
$b' \leftarrow \mathcal{A}^{|O_b\rangle, \mathsf{Reprogram}}$    $O_1 := O_1^{x_1||x_2 \rightarrow y}$
return $b'$                           return $x_1$
```

Fig. 2. Adaptive reprogramming games for $b \in \{0, 1\}$.

Lemma 5. *(Grover's Algorithm. [22]) Let $F : X \rightarrow \{0, 1\}$ be a predicate mapping an element of set X to a bit and $F^{-1}(1) = \{x : F(x) = 1\}$ is non-empty. Let $t = |F^{-1}(1)| > 0$. There is a quantum algorithm that randomly returns $x^* \in F^{-1}(1)$ with at most $O(\sqrt{\frac{|X|}{t}})$ quantum queries to F.*

We call the above quantum algorithm Grover's algorithm.

3 Subset Resilience and Its Variants

Subset resilience is first proposed in HORS [31], a few-time HBS scheme. In this section, we give definitions of several variants and analyze their generic security.

3.1 Definitions

Definition 3. *(Subset Cover. [31, 33]) Let $H = (h_1, ...h_k)$ be a hash function mapping $\{0, 1\}^m$ to $[t]^k$ and $r \geq 0$ be an integer. We say that $(r + 1)$-tuple $(x, x_1, ..., x_r) \in \{0, 1\}^{m(r+1)}$ is an (r, k)-subset cover of H if it holds that $\{h_i(x)\}_{i \in [k]} \subset \{h_i(x_j)\}_{i \in [k], j \in [r]}$ and $x \notin \{x_j\}_{j \in [r]}$.*

Definition 4. *(Restricted Subset Cover. [33]) Let $H = (h_1, ...h_k)$ be a hash function mapping $\{0, 1\}^m$ to $[t]^k$ and $r \geq 0$ be an integer. We say that $(r + 1)$-tuple $(x, x_1, ..., x_r) \in \{0, 1\}^{m(r+1)}$ is an (r, k)-restricted subset cover of H if it holds that $x \notin \{x_j\}_{j \in [r]}$, and for $\forall i \in [k]$, $h_i(x) \in \{h_i(x_j)\}_{j \in [r]}$.*

Next, we show a weaker statement called *weak subset cover* that is useful in the following sections.

Definition 5. *(Weak Subset Cover.) Let $H = (h_1, ...h_k)$ be a hash function mapping $\{0,1\}^m$ to $[t]^k$ and $r \geq 0$ be an integer. We say an $(r+1)$-tuple $(x_1, ..., x_{r+1}) \in \{0,1\}^{m(r+1)}$ is an (r,k)-weak subset cover of H if for $\forall i \in [k]$, it holds that $|\{h_i(x_j)\}_{j \in [r+1]}| \leq r$ and x_j's are distinct. In other words, there exists a collision in $x_1, ..., x_{r+1}$ w.r.t. each h_i.*

Corollary 2. *If $(x, x_1, ..., x_r)$ is an (r,k)-restricted subset cover of H, it is also an (r,k)-weak subset cover of H.*

If it is hard for any polynomial-time adversary to find a (restricted/weak-) subset cover, then we say that the hash function family is (restricted/weak-) subset-resilient.

Definition 6. *Let $\mathcal{H} = \{H : \{0,1\}^m \to [t]^k\}$ be a hash function family. Let $\mathcal{A}$ be an adversary that takes as input $H = (h_1, ..., h_k) \leftarrow \mathcal{H}$, runs at most q hash queries and finally outputs $(x, x_1, ..., x_r) \in \{0,1\}^{m(r+1)}$. Define*

$$Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{SR}}(\mathcal{A}) \triangleq \Pr_{\mathcal{H},\mathcal{A}}\left[\{h_i(x)\}_{i\in[k]} \subset \{h_i(x_j)\}_{i\in[k],j\in[r]} \wedge x \notin \{x_j\}_{j\in[r]}\right]$$

and

$$Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{rSR}}(\mathcal{A}) \triangleq \Pr_{\mathcal{H},\mathcal{A}}\left[\forall i \in [k], h_i(x) \in \{h_i(x_j)\}_{j\in[r]} \wedge x \notin \{x_j\}_{j\in[r]}\right].$$

Let $\mathcal{A}$ be an adversary that takes as input $H = (h_1, ..., h_k) \leftarrow \mathcal{H}$, runs at most q hash queries and finally outputs $(x_1, ..., x_{r+1}) \in \{0,1\}^{m(r+1)}$. Define

$$Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{wSR}}(\mathcal{A}) \triangleq \Pr_{\mathcal{H},\mathcal{A}}\left[\forall i \in [k], \left|\{h_i(x_j)\}_{j\in[r+1]}\right| \leq r \wedge x_1, x_2, ..., x_{r+1} \text{ are distinct }\right].$$

We say that $\mathcal{H}$ is a secure (r,k)(-restricted/weak) subset resilient hash function family or (r,k)-SRH(/rSRH/wSRH) if $Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{SR}}(\mathcal{A})/Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{rSR}}(\mathcal{A})/$ $Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{wSR}}(\mathcal{A})$ is negligible for any probabilistic polynomial-time quantum adversary $\mathcal{A}$.

3.2 Generic Security with Quantum Queries

Theorem 1. *Let $\mathcal{H} = \{H : \{0,1\}^m \to [t]^k\}$ be a random function family and r be a positive integer. For any quantum probabilistic polynomial-time adversary $\mathcal{A}$, it holds that*

$$Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{wSR}}(\mathcal{A}) \leq (2q+1)^{2(r+1)}\left(\frac{r^2}{t}\right)^k, \tag{3}$$

$$Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{rSR}}(\mathcal{A}) \leq (2q+1)^{2(r+1)}(2r+2)\left(\frac{r}{t}\right)^k, \tag{4}$$

and

$$Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{SR}}(\mathcal{A}) \leq (2q+1)^{2(r+1)}(2r+2)\left(\frac{rk}{t}\right)^k. \tag{5}$$

Proof. Since $\mathcal{H}$ is a random function family, the success probability of adversaries can be evaluated by Lemma 1. For $i \in [k]$, denote by $y^{(i)} \in [t]$ the i-th element of y.

1. (Proof of (3).)
For $H = (h_1, ..., h_k) : \{0,1\}^* \to [t]^k$, define $R_1 \subseteq [t]^{k(r+1)}$ as follows:

$$R_1 \triangleq \{(y_1, y_2, ..., y_{r+1}) : \forall i \in [k], \left|\{y_j^{(i)}\}_{j\in[r+1]}\right| \leq r\}.$$

We analyze the size of R_1. From $(y_1, ..., y_{r+1}) \in R_1$, for every $i \in [k]$, (at least) two of $y_1^{(i)}, ..., y_{r+1}^{(i)}$ are equal. Fix an $i \in [k]$, we can traverse all the possible $(y_1^{(i)}, ..., y_{r+1}^{(i)})$ w.r.t. i as follows:
(a) Pick a pair of indices a_1, a_2 from $[r]$.
(b) Pick $y \in [t]$, let $y_{a_1} = y_{a_2} = y$.
(c) Traverse all possible values of $y_j^{(i)}$ for all $j \notin \{a_1, a_2\}$ and $j \in [r+1]$.
The numbers of choices in the three steps are $\binom{r+1}{2}$, t and t^{r-1} respectively. Thus, for all $i \in [k]$, the total number of possible values of $(y_1, ..., y_{r+1}) \in R_1$ is at most

$$\left(\binom{r+1}{2} \cdot t \cdot t^{r-1}\right)^k = \left(\frac{(r+1)r}{2}t^r\right)^k \leq (r^2t^r)^k.$$

In addition, it is not hard to see that relation R_1 is not ordered (which means that for any $\pi \in \mathsf{Perm}([r+1])$, the statement $(y_1, ..., y_{r+1}) \in R_1$ is equivalent to the statement $(y_{\pi(1)}, ..., y_{\pi(r+1)}) \in R_1$). Due to Lemma 1, we have

$$\mathrm{Adv}_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{wSR}}(\mathcal{A}) \leq (2q+1)^{2(r+1)}\frac{(r^2t^r)^k}{t^{(r+1)k}} = (2q+1)^{2(r+1)}\left(\frac{r^2}{t}\right)^k,$$

which is what we expected.
2. (Proof of (4).)
Note that in Lemma 1, the elements in a solution have to be distinct, but those in a restricted subset cover do not. (In a restricted subset cover, only $x \notin \{x_j\}_{j\in[r]}$ is demanded, and thus x_j can be equal to another $x_{j'}$.) We divide a restricted subset cover into several cases.
Fix r, $\mathcal{H}$ and $\mathcal{A}$. Recall that $H = (h_1, ..., h_k) \leftarrow \mathcal{H}$ and $(x, x_1, ..., x_r) \leftarrow \mathcal{A}(H)$. Let $1 \leq s \leq r$ be some integer. Define

$$f(s) \triangleq \Pr_{\mathcal{H},\mathcal{A}}\left[\forall i \in [k], h_i(x) \in \{h_i(x_j)\}_{j\in[r]} \wedge x \notin \{x_j\}_{j\in[r]} \wedge \left|\{x_j\}_{j\in[r]}\right| = s\right].$$

Then, we have

$$\mathrm{Adv}_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{rSR}}(\mathcal{A}) = \sum_{s\in[r]} f(s),$$

and

$$f(r) = \Pr_{\mathcal{H},\mathcal{A}}\left[\{h_i(x)\}_{i\in[k]} \subset \{h_i(x_j)\}_{i\in[k],j\in[r]} \wedge x, x_1, ..., x_r \text{ are distinct}\right].$$

First, we give a bound for the case that $r = s$.

Lemma 6. $f(r) \leq (2q+1)^{2(r+1)}(r+1)(\frac{r}{t})^k$.

Proof. For $H = (h_1, ..., h_k) : \{0,1\}^* \to [t]^k$, define $R_2 \subseteq [t]^{k(r+1)}$ as follows:

$$R_2 \triangleq \big\{(y, y_1, ..., y_r) : \forall i \in [k], y^{(i)} \in \{y_j^{(i)}\}_{j\in[r]}\big\}.$$

Next, we analyze the size of R_2. For convenience, we call the first element of $(y, y_1, ..., y_r) \in R_2$ as y_0.

First, there are exactly t^k possible values of y_0. Then, for any fixed $y_0 = (y_0^{(1)}, ..., y_0^{(k)})$, it holds that $y_0^{(i)} \in \{y_j^{(i)}\}_{j\in[r]}$ for each $i \in [k]$. This implies that $y_j^{(i)} = y_0^{(i)}$ for some $j \in [r]$. We can traverse all the possible value of $(y_1^{(i)}, ..., y_r^{(i)})$ for each i w.r.t. y_0 by the following steps:

(a) Pick $j \in [r]$ and let $y_j^{(i)} = y_0^{(i)}$.
(b) Traversing all the possible value of $y_{j'}^{(i)}$ for all $j' \in [r]$ and $j' \neq j$.

The number of possible values of $(y_1^{(i)}, ..., y_r^{(i)})$ w.r.t. y_0 is at most $r \cdot t^{r-1}$ for each i. Thus, considering all $i \in [k]$ and traversing all possible values of y_0, the total number of $(y_0, y_1, ..., y_r)$ is at most

$$(r \cdot t^{r-1})^k \cdot t^k = (rt^r)^k.$$

Unlike R_1, relation R_2 is ordered. Define

$$R_2^* \triangleq \{(y_1, ..., y_{r+1}) : \exists \pi \in \mathsf{Perm}([r+1]) \text{ s.t. } (y_{\pi(1)}, ..., y_{\pi(r+1)}) \in R_2\}.$$

Observe that for any $\pi \in \mathsf{Perm}([r])$, the statement $(y, y_1, ..., y_r) \in R_2$ is equivalent to the statement $(y, y_{(\pi(1))}, ..., y_{\pi(r)}) \in R_2$. This implies that the order of R_2 is only determined by the first element. Thus, we can traverse all the possible values of $(y_1, ..., y_{r+1}) \in R_2^*$ by the following steps:

(a) Pick $(y, y_1, ..., y_r) \in R_2$.
(b) Pick $j \in [r+1]$, and insert y between $(j-1)$-th element and the j-th element of $(y_1, ..., y_r)$. In other words, traverse $(y, y_1, ...y_r)$, $(y_1, y, y_2, ..., y_r)$, ..., $(y_1, ..., y_r, y)$.

Thus, we have

$$|R_2^*| \leq (r+1)|R_2| \leq (r+1)(rt^r)^k.$$

Due to Lemma 1, we have

$$f(r) \leq (2q+1)^{2(r+1)} \frac{(r+1)(rt^r)^k}{t^{(r+1)k}} = (2q+1)^{2(r+1)}(r+1)\left(\frac{r}{t}\right)^k.$$

Next, we consider the case that $s < r$.

If the adversary output an (r,k)-restricted subset cover $(x, x_1, ..., x_r)$ such that $|\{x_j\}_{j\in[r]}| = s < r$. Let $\{x_j\}_{j\in[r]} = \{x'_{j'}\}_{j'\in[s]}$ after reordering. Then, it is not hard to see that $(x, x'_1, ..., x'_s)$ is an (s,k)-restricted cover and all the elements are distinct. The probability of this event is also bounded by Lemma 6. That is, for all $1 \leq s \leq r$ it holds that

$$f(s) \leq (2q+1)^{2(s+1)}(s+1)\left(\frac{s}{t}\right)^k \leq (2q+1)^{2(s+1)}(r+1)\left(\frac{r}{t}\right)^k.$$

Thus, we have

$$\mathrm{Adv}_{\mathcal{H},q}^{(r,k)\text{-rSR}}(\mathcal{A}) = \sum_{s\in[r]} (2q+1)^{2(s+1)}(r+1)\left(\frac{r}{t}\right)^k \leq (2q+1)^{2(r+1)}(2r+2)\left(\frac{r}{t}\right)^k.$$

3. (Proof of (5).)
   Similarly, fix $r, \mathcal{H}, \mathcal{A}$. For $s \in [r]$, we define

$$g(s) \triangleq \Pr_{\mathcal{H},\mathcal{A}}\left[x \notin \{x_1, ..., x_r\} \wedge \{h_i(x)\}_{i\in[k]} \subset \{h_i(x_j)\}_{i\in[k],j\in[r]} \wedge \left|\{x_j\}_{j\in[r]}\right| = s\right],$$

and thus

$$\mathrm{Adv}_{\mathcal{H},q}^{(r,k)\text{-SR}}(\mathcal{A}) = \sum_{s\in[r]} g(s).$$

As above, we first consider the case that $s = r$.
For $H = (h_1, ..., h_k) : \{0,1\}^* \to [t]^k$, define $R_3 \subseteq [t]^{k(r+1)}$ as follows:

$$R_3 \triangleq \{(y, y_1, ..., y_r) : \{y^{(i)}\}_{i\in[k]} \subseteq \{y_j^{(i)}\}_{i\in[k],j\in[r]}\}.$$

We divide R_3 into k subsets $R_{3,1}, ..., R_{3,k}$, where, for $m \in [k]$,

$$R_{3,m} \triangleq \{((y, y_1, ..., y_r) \in R_3 : \left|\{y^{(i)}\}_{i\in[k]}\right| = m\}.$$

Observe that $R_{3,m}$'s are disjoint and that $R_3 = \bigcup_{m\in[k]} R_{3,m}$. More precisely, the statement $(y, y_1, ..., y_r) \in R_{3,m}$ implies that $\{y^{(i)}\}_{i\in[k]}$ contains exactly m elements in $[t]$, and $\{y_j^{(i)}\}_{i\in[k],j\in[r]}$ covers them. Since there are at most rk elements in set $\{y_j^{(i)}\}_{i\in[k],j\in[r]}$, there are rk "chances" to cover the m target elements. We can traverse all the elements of $R_{3,m}$ by the following steps:

(a) Pick m distinct elements $x_1, ..., x_m$ from $[t]$. Let $X = \{x_1, ..., x_m\}$. The number of choices in this step is $\binom{t}{m}$.

(b) Pick $y^{(1)}, ..., y^{(k)}$ from X^k such that $\{y^{(i)}\}_{i\in[k]} = X$.
This step is equivalent to the experiment of putting k different balls into m different bins such that there is at least one ball in each bin. The number of choices is $\left\{ {k \atop m} \right\} \cdot m!$, where $\left\{ {k \atop m} \right\}$ denotes Stirling number of the second kind.

(c) Next, we require that $\{y_j^{(i)}\}_{i\in[k],j\in[r]}$ covers $X = \{y^{(i)}\}_{i\in[k]}$. Since $|X| = m$, we only need to choose m elements of $\{y_j^{(i)}\}_{i\in[k],j\in[r]}$, make them equal to a permutation of X, and do not have any demand for the remaining $(rk - m)$ elements.
The number of choices in the two substeps are $\binom{rk}{m}$ and $m!$ respectively.

(d) Finally, since the remaining $(rk - m)$ elements have no demand, traverse all the possible $y_j^{(i)}$ that have not been assigned in the above steps.
The number of choices in this step is $t^{(rk-m)}$.

To sum up, the total number of elements in $R_{3,m}$ is

$$\begin{aligned}|R_{3,m}| =& \binom{t}{m} \left\{ {k \atop m} \right\} \cdot m! \cdot \binom{rk}{m} \cdot m! \cdot t^{rk-m} \\ \leq& \frac{t^m}{m!} \cdot \left\{ {k \atop m} \right\} \cdot m! \cdot \binom{rk}{m} \cdot m! \cdot t^{rk-m} \\ =& \left\{ {k \atop m} \right\} \cdot \binom{rk}{m} \cdot m! \cdot t^{rk} \\ =& \left\{ {k \atop m} \right\} \cdot (rk)_m \cdot t^{rk},\end{aligned}$$

where $(\cdot)_m$ denotes the falling factorial:

$$(x)_m = x \cdot (x-1) \cdot ... \cdot (x - m + 1).$$

Thus, we have

$$|R_3| = \sum_{m=1}^{k} |R_{3,m}| \leq \sum_{m=1}^{k} \left\{ {k \atop m} \right\} \cdot (rk)_m \cdot t^{rk} = (rk)^k \cdot t^{rk},$$

where the last equality uses the fact that $\sum_{m=1}^{k} \left\{ {k \atop m} \right\} (x)_m = x^k$.
Similar to R_2, we define

$$R_3^* \triangleq \{(y_1, ..., y_{r+1}) : \exists \pi \in \mathsf{Perm}([r+1]) \text{ s.t. } (y_{\pi(1)}, ..., y_{\pi(r+1)}) \in R_3\},$$

and then we have

$$|R_3^*| \leq (r+1)|R_3| \leq (r+1)(rk)^k \cdot t^{rk}.$$

Due to Lemma 1, we have

$$g(r) \leq (2q+1)^{2(r+1)} \frac{(r+1)(rk)^k \cdot t^{rk}}{t^{(r+1)k}} = (2q+1)^{2(r+1)}(r+1)\left(\frac{rk}{t}\right)^k,$$

and for $s \in [r]$,

$$g(s) \leq (2q+1)^{2(s+1)}(s+1)\left(\frac{sk}{t}\right)^k \leq (2q+1)^{2(s+1)}(r+1)\left(\frac{rk}{t}\right)^k.$$

Thus,

$$\mathrm{Adv}_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{SR}}(\mathcal{A}) = \sum_{s\in[r]}(2q+1)^{2(s+1)}(r+1)\left(\frac{rk}{t}\right)^k \leq (2q+1)^{2(r+1)}(2r+2)\left(\frac{rk}{t}\right)^k.$$

3.3 Target Subset Resilience

Target subset resilience (TSR) [31] is a variant of subset resilience. In (r,k)-TSR experiment, the adversary is given a hash function $H = (h_1, ..., h_k)$ and r random targets $x_1, ...x_r$. Then, the adversary is required to output a single element x such that $\{h_i(x)\}_{i\in[k]} \subset \{h_i(x_j)\}_{i\in[k],j\in[r]}$. It is not hard to see that (r,k)-TSR is a weaker notion than (r,k)-SR.

In this section we propose a target version of restricted subset resilience, which is called *extended target subset resilience* (eTSR). Unlike TSR, the adversary in eTSR can adaptively control the target to some extent. In detail, the adversary is able to adaptively query a (classical) oracle Box. For a query x_j, Box randomly chooses $z_j \in \{0,1\}^n$ and returns $(z_j, H(z_j||x_j))$. After r queries, the adversary is required to output (x,z) such that for each $i \in [k]$, $h_i(z||x) \in \{h_i(z_j||x_j)\}_{j\in[r]}$ and $(x,z) \notin \{(x_j,z_j)\}_{j\in[r]}$ hold. Note that (r,k)-eTSR is a weaker notion than than (r,k)-rSR.

Definition 7. *(Extended Target Subset Resilience.) Let $\mathcal{H} = \{H = (h_1, ..., h_k) : \{0,1\}^{m+n} \to [t]^k\}$ be a hash function family. Let $\mathcal{A}^{\mathsf{Box}}$ be an adversary that queries Box at most r times, computes H at most q times and then outputs $(x,z) \in \{0,1\}^{m+n}$. Define*

$$Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{eTSR}}(\mathcal{A}) \triangleq \Pr_{\mathsf{Box},\mathcal{H},\mathcal{A}}\left[\forall i \in [k], h_i(z||x) \in \{h_i(z_j||x_j)\}_{j\in[r]} \wedge (x,z) \notin \{(x_j,z_j)\}_{j\in[r]}\right].$$

We say that hash function family $\mathcal{H}$ is an (r,k)-extended target-subset-resilient hash function family ((r,k)-eTSRH) if $Adv_{\mathcal{H},q}^{(r,k)\text{-}\mathsf{eTSR}}(\mathcal{A})$ is negligible for any probabilistic polynomial-time quantum adversary $\mathcal{A}$.

Here we model $\mathcal{H}$ as a quantum-accessible $\mathsf{H} : \{0,1\}^{m+n} \to [t]^k$ and $\mathsf{h}_1, ..., \mathsf{h}_k : \{0,1\}^{m+n} \to [t]$ be the partial oracle. Then, the experiment of eTSR is depicted as **Game 0** in Fig. 3.

Theorem 2. *Let $\mathcal{H}$ be modeled as a quantum-accessible random oracle H and r be a positive integer. For any quantum probabilistic polynomial-time adversary $\mathcal{A}$, it holds that*

$$Adv_{\mathsf{H},q}^{(r,k)\text{-}\mathsf{eTSR}}(\mathcal{A}) \leq \frac{3r}{2}\sqrt{\frac{q+r+1}{2^n}} + 8(q+r+2)^2\left(\frac{r}{t}\right)^k.$$

```
Game 0
$(x, z) \leftarrow \mathcal{A}^{|\mathsf{H}\rangle,\mathsf{Box}}()$
if $\forall i \in [k], \mathsf{h}_i(z||x) \in \{\mathsf{h}_i(z_j||x_j)\}_{j\in[r]}$
    if $(x, z) \notin \{(x_j, z_j)\}_{j\in[r]}$ return 1
return 0
```

```
Box($x_j$)
$z_j \leftarrow \{0, 1\}^n$
return $(z_j, \mathsf{H}(z_j||x_j))$
```

```
Game 1
$(x, z) \leftarrow \mathcal{A}^{|\mathsf{H}\rangle,\mathsf{Box'}}()$
if $\forall i \in [k], \mathsf{h}_i(z||x) \in \{y_j^{(i)}\}_{j\in[r]}$
    if $(x, z) \notin \{(x_j, z_j)\}_{j\in[r]}$ return 1
return 0
```

```
Box'($x_j$)
$z_j \leftarrow \{0, 1\}^n$
$y_j := y_j^{(1)}||...||y_j^{(k)} \leftarrow [t]^k$
$\mathsf{H} := \mathsf{H}^{z_j||x_j \to y_j}$
return $(z_j, y_j)$
```

Fig. 3. Hybrid arguments in the proof of Theorem 2.

Proof. We use the technique in the security proof of (multi-target) extended collision resistance in [21,27]. We prove this theorem by showing the following games:

- **Game 0** is the original experiment of eTSR.
- **Game 1** differs from **Game 0** in that Box is replaced by $\mathsf{Box'}$. Every time $\mathcal{A}$ queries x_j to the oracle $\mathsf{Box'}$, $\mathsf{Box'}$ randomly picks z_j, randomly chooses $y_j^{(1)}, ..., y_j^{(k)} \in [t]$ and reprograms $\mathsf{h}_i(z_j||x_j) := y_j^{(i)}$ for each $i \in [k]$. (In other words, it reprograms $\mathsf{H}(z_j||x_j) := y_j = y_j^{(1)}||...||y_j^{(k)}$.) Then, it returns (z_j, y_j) to the adversary. See details in Fig. 3.

  Next, we show that the probabilities of **Game 0** and **Game 1** are negligible close by Lemma 4. If not, the adversary $\mathcal{A}$ can be used to distinguish Repro_0 and Repro_1 in Fig. 2. In **Game 0**, H is simulated by O_0 and Box is simulated by $\mathsf{Reprogram}_0$ with additional classical query to O_0. In **Game 1**, $\mathsf{Box'}$ is simulated by $\mathsf{Reprogram}_1$ with additional classical query to O_1. In total, it issues $(q+r+1)$ queries to O_b (q for simulating H, r for simulating $\mathsf{Box}/\mathsf{Box'}$ and 1 for the final verification). Due to Lemma 4, we have

$$|\Pr[\textbf{Game 0}] - \Pr[\textbf{Game 1}]| \leq \frac{3r}{2}\sqrt{\frac{q+r+1}{2^n}}. \tag{6}$$

- In **Game 1**, the adversary outputs (x, z) such that for all $i \in [k]$, $\mathsf{h}_i(z||x)$ is covered by $\{y_j^{(i)}\}_{j\in[r]}$. Define $S = \{y_{a_1}^{(1)}||...||y_{a_k}^{(k)}\}_{(a_1,...,a_k)\in[r]^k}$. In other words, S contains all $y = y^{(1)}||...||y^{(k)}$ where $y^{(i)} \in \{y_j^{(i)}\}_{j\in[r]}$ for each i. Thus, the adversary is to output (x, z) such that $H(z||x) \in S$ and (x, z) is not equal to any (x_j, z_j).

  Note that $|S| \leq r^k$. Without loss of generality, we suppose $|S| = r^k$. (If $|S| < r^k$, the success probability is obviously smaller. Here our purpose is to find an upper bound of the probability.) Reorder $S = \{y'_1, ..., y'_{r^k}\}$.

Next, we use an adversary succeeding in **Game 1** to construct a reduction breaking $\mathsf{Avg\text{-}Search}_\lambda$ in Lemma 3.

Let $f \leftarrow D_\lambda : \{0,1\}^{m+n} \to \{0,1\}$ and $\lambda = (\frac{r}{t})^k$. Let $I : \{0,1\}^{m+n} \to [r^k]$ and $g : \{0,1\}^{m+n} \to [t]^k \backslash S$ be random functions. Construct $\tilde{\mathsf{H}} : \{0,1\}^{m+n} \to [t]^k$ as follows: for any $(z||x) \in \{0,1\}^{m+n}$, define:

$$\tilde{\mathsf{H}}(z||x) = \begin{cases} y_j, & x = x_j \wedge z = z_j, \\ y'_{I(z||x)}, & f(z||x) = 1, \\ g(x), & otherwise. \end{cases}$$

Note that the outputs of $\tilde{\mathsf{H}}$ distributes uniformly. Thus, an adversary in **Game 2** finds (x, z) such that $f(z||x) = 1$. Due to Lemma 3, we have

$$\Pr[\textbf{Game 1}] \leq 8(q + r + 1 + 1)^2 \left(\frac{r}{t}\right)^k = 8(q + r + 2) \left(\frac{r}{t}\right)^k. \tag{7}$$

From Eq. (6) and (7), we complete the proof.

4 Hash-Based Signature Schemes

In this section, we introduce two practical stateless HBS schemes, SPHINCS [7] and SPHINCS+ [8]. Due to the complicated constructions, we follow the algorithmic descriptions in [16] by treating the building blocks as black boxes. Briefly speaking, SPHINCS(+) introduces a stateful HBS with hypertrees (denoted by HT) and a few-time HBS. We first introduce the two building blocks and then the schemes.

4.1 Building Blocks – Few-Time HBS Schemes

In this section, we introduce few-time HBS schemes, which are used as building blocks in SPHINCS and SPHINCS+.

The first is Hash to Obtain Random Subsets (HORS) [31]. The outline of HORS is as follows. In the key generation algorithm, it picks a one-way function $f : \{0,1\}^{l(n)} \to \{0,1\}^n$ and an (r, k)-subset-resilient function $H = (h_1, ..., h_k) : \{0,1\}^m \to [t]^k$. Then, it picks t random strings $s_1, ..., s_t$ from $\{0,1\}^{l(n)}$ and computes $y_j = f(s_j)$ for each $j \in [t]$. Let $(s_1, ..., s_t)$ be the secret key and $(y_1, ..., y_t)$ be the public key. In the signing algorithm, it reveals k elements from $\{s_j\}_{j \in [t]}$ determined by $H(m)$. Due to (r, k)-subset resilience of H, it is hard to find a message m^* such that the secret values in the corresponding signature are covered by r (classical) queries to the signing oracle. The formal description is depicted in Fig. 4.

SPHINCS [7] introduces a variant of HORS called HORST (HORS with trees). HORST compresses the public key with a (bitmarked) hash tree, and thus the signature needs to contain the corresponding authentication path. This

```
$\mathsf{HORS.KeyGen}(1^\lambda)$

$F \leftarrow \mathcal{F}_n$, $H = (h_1, ..., h_k) \leftarrow \mathcal{H}$.
**for** $j \in [t]$, $s_j \leftarrow \{0,1\}^{l(n)}$, $y_j = f(s_j)$
$Y = (y_1, ..., y_t)$, $S = (s_1, ..., s_t)$
$pk = (Y, f, H)$, $sk = (S, f, H)$.
**return** $(pk, sk)$.

$\mathsf{HORS.Sig}(sk, m)$

Parse $S = (s_1, ..., s_t)$ and $H = (h_1, ..., h_k)$
**for** $i \in [k]$, $x_i = s_{h_i(m)}$.
**return** $\sigma = (x_1, ..., x_k)$.

$\mathsf{HORS.Ver}(pk, m, \sigma)$

Parse $\sigma = (x_1, ..., x_k)$ and $H = (h_1, ..., h_k)$.
**if for** $i \in [k]$, $y_{h_i(m)} = f(x_i)$ **return** 1
**return** 0
```

Fig. 4. Construction of Hash to Obtain Subsets (HORS).

operation does not hurt the security except that it requires a second-preimage-resistant hash function.

Furthermore, SPHINCS+ [8] introduces an improvement of HORST, which is called FORS (Forest of Random Subsets). The main differences between HORST and FORS are as follows. First, the key generation algorithm picks kt random strings from $\{0,1\}^{l(n)}$ (rather than t strings) and divides them into k groups of t strings. In the signing algorithm, instead of revealing k elements from t strings, FORS reveals *one* element from each group. Second, FORS uses a tweakable hash function **F** instead of the one-way function f, where the tweaks are the indices of the strings. Third, instead of using bitmarked hash functions in the hash tree, FORS uses a tweakable hash function **Th** in generating hash trees, where the tweaks are the addresses of the nodes. Finally, since there are k hash trees in FORS, it compresses the k roots by calling **Th** and denotes the value as the public key[4].

In FORS, the message is not hashed. The signing algorithm directly splits the message m into k digits with size $\log t$ and then proceeds with the following steps. Thus, the scheme is not EUF-CMA secure. (One can forge a signature on message $m_1||m_2$ given signatures on $m_1||m_2^*$ and $m_1^*||m_2$.) In practice, FORS has to be used on hashed messages to achieve EUF-CMA. We call FORS with integrated hashing as *simplified* FORS (sFORS).

[4] See formal definitions of tweakable hash functions and hash trees in the full version [34].

```
sFORS.KeyGen($1^\lambda$)
    $H = (h_1, ..., h_k) \leftarrow \mathcal{H}$
    for $(i, j) \in [k] \times [t]$, $s_{i,j} \leftarrow \{0,1\}^{l(n)}$, $y_{i,j} = \mathbf{F}((i, j), s_{i,j})$
    for $i \in [k]$, $y_i \leftarrow \mathsf{TreeGen}(\mathbf{Th}, t, (y_{i,1}, ..., y_{i,t}))$.
    $y_0 = \mathbf{Th}(0, (y_1, ...y_k))$, $S = (s_{1,1}, ..., s_{k,t})$
    $pk = (y_0, H)$, $sk = (S, H)$.
    return $(pk, sk)$.

sFORS.Sig($sk, m$)
    Parse $S = (s_{1,1}, ..., s_{k,t})$ and $H = (h_1, ..., h_k)$
    for $i \in [k]$, $x_i = s_{i,h_i(m)}$.
    for $(i, j) \in [k] \times [t]$, $y_{i,j} = \mathbf{F}((i, j), s_{i,j})$.
    for $i \in [k]$, $\pi_i \leftarrow \mathsf{TreeProv}(\mathbf{Th}, t, (y_{i,1}, ..., y_{i,t}), h_i(m))$
    return $\sigma = (x_1, ..., x_k, \pi_1, ..., \pi_k)$.

sFORS.pkFromSig($m, \sigma, H$)
    Parse $\sigma = (x_1, ..., x_k, \pi_1, ..., \pi_k)$ and $H = (h_1, ..., h_k)$.
    for $i \in [k]$, $y_i \leftarrow \mathsf{TreeVer}(\mathbf{Th}, t, h_i(m), \mathbf{F}((i, j), x_i), \pi_i)$.
    return $y'_0 = \mathbf{Th}(0, (y_1, ..., y_k))$

sFORS.Ver($pk, m, \sigma$)
    Parse $pk = (y_0, H)$
    return $[\![\mathsf{sFORS.pkFromSig}(m, \sigma, H) = y_0]\!]$
```

Fig. 5. Construction of Simplified FORS.

In SPHINCS+ [8], sFORS is never directly used. In each signing operation, it introduces a (pseudo-)randomizer z. The message is then hashed with z, and z is included in the signature. It results in a new scheme that achieves higher bit security. We call it *randomized* FORS (rFORS). The formal constructions are depicted in Fig. 5 and 6.

```
$\mathsf{rFORS.KeyGen}(1^\lambda)$
$(pk, sk) \leftarrow \mathsf{sFORS.KeyGen}(1^\lambda)$
$k \leftarrow \{0,1\}^n$
return $(pk, (sk, k))$.

$\mathsf{rFORS.Sig}((sk, k), m)$
$z = \mathsf{PRF}(k, m)$
$\sigma \leftarrow \mathsf{sFORS.Sig}(sk', z||m)$
return $(z, \sigma)$.

$\mathsf{rFORS.pkFromSig}(m, (z, \sigma), H)$
return $\mathsf{sFORS.pkFromSig}(z||m, \sigma, H)$

$\mathsf{rFORS.Ver}(pk, m, (z, \sigma))$
Parse $pk = (y_0, H)$
return $[\![\mathsf{rFORS.pkFromSig}(m, (z, \sigma), H) = y_0]\!]$
```

Fig. 6. Construction of Randomized FORS.

4.2 Building Block – HT: The hypertree

HT is another building block in SPHINCS-like structure. It behaves as a stateful signature scheme, where the signing and verification algorithms additionally take as input a state $\mathsf{st} \in ST$. Note that the state space is of polynomial size. The syntax is defined as follows.

Definition 8. *A hyper tree signature scheme $\mathsf{HT} = (\mathsf{HT.KeyGen}, \mathsf{HT.Sign}, \mathsf{HT.Ver})$ consists of three polynomial-time algorithms along with an associated message space $\mathcal{M} = \{\mathcal{M}_n\}$ and a state space ST such that:*

- *The key generation algorithm KeyGen takes as input the security parameter 1^n. It outputs a pair of keys (pk, sk).*
- *For security parameter n, the signing algorithm Sign takes as input a secret key sk, a message $m \in \mathcal{M}_n$ and a state $\mathsf{st} \in ST$. It outputs a signature σ.*
- *For security parameter n, the verification algorithm Ver takes as input a public key pk, a message $m \in \mathcal{M}_n$, a signature σ and a state $\mathsf{st} \in ST$. It outputs a bit b.*

For any $(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$, $m \in \mathcal{M}_n$, $\mathsf{st} \in ST$ and $\sigma \leftarrow \mathsf{Sign}(sk, m, \mathsf{st})$, it holds that $\mathsf{Ver}(pk, m, \sigma, \mathsf{st}) = 1$.

Although SPHINCS and SPHINCS+ use different HT, the security notions for HT are the same and implicitly contained in [7,8]. The security is a stateful version of existential unforgeability under non-adaptive chosen message attacks.

We call it *existential unforgeability under non-adaptive chosen message attacks with states* (EUF-sNACMA). In detail, the security experiment is defined as follows.

```
**Experiment** $\mathsf{Exp}^{\mathsf{EUF\text{-}sNACMA}}_{\mathsf{HT},q_s,q_H}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$
    $(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$
    $(\{(m_i, \mathsf{st}_i)\}_{i\in[q_s]}, S) \leftarrow \mathcal{A}_1()$
    If $\mathsf{st}_i$ are not distinct, **return** 0
    **For** $i \in [q_s]$, $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i, \mathsf{st}_i)$
    $(m^*, \sigma^*, \mathsf{st}^*) \leftarrow \mathcal{A}_2(pk, S, \{\sigma_i\}_{i\in[q_s]})$
    If $\mathsf{st}^* = \mathsf{st}_j$ for some $j \wedge m^* \neq m_j \wedge \mathsf{Ver}(pk, m^*, \sigma^*, \mathsf{st}^*) = 1$, **return** 1,
otherwise **return** 0.
```

Remark 2. In this paper, we do not depict the detailed construction of HT in SPHINCS or SPHINCS+, since we always use it as a black box. We only care about the security.

4.3 SPHINCS and SPHINCS+

SPHINCS and SPHINCS+ are practical stateless HBS schemes built from the above two blocks. In each signing execution, the signer first pseudorandomly picks a state from ST in HT, which authenticates the public key of the few-time HBS. A signature contains (1) a few-time signature of the message, (2) an HT signature of the public key of the corresponding few-time HBS, and (3) a (pseudo-)randomizer. Note that in SPHINCS and its variants, the few-time signatures are never explicitly verified. Instead, it uses pkFromSig to recover the public key from the message and signature. The security of HT guarantees that only the real public key can be verified in the next steps.

In SPHINCS, the few-time signature scheme is HORST (HORS with trees), a variant of HORS. It compresses the HORS public key by a Merkle tree structure, and the compressed public key can be generated by an algorithm pkFromSig from the message and the signature. In this section, we also use HORS.Sig and HORS.pkFromSig to denote the algorithms of HORS with trees.

The outlines of SPHINCS and SPHINCS+ are depicted in Fig. 7. The main differences are the choices of few-time signature schemes and the ways to pick the index. In particular, the index of SPHINCS+ is calculated by a hash value of the message and the randomizer. Thus, it is not directly contained in the signature.

Remark 3. The scheme in Fig. 7 is the deterministic version of SPHINCS+. It can be converted into a probabilistic version by adding a random salt to the input of $\mathsf{PRF}_{\mathsf{msg}}$. In this paper, we mainly focus on the deterministic version and will discuss the probabilistic version in Sect. 6.2.

```
$\mathsf{SPHINCS.KeyGen}(1^\lambda)$
$sk_{\mathsf{seed}} \leftarrow \{0,1\}^n$,
$(pk_{\mathsf{HT}}, sk_{\mathsf{HT}}) \leftarrow \mathsf{HT.KeyGen}(1^\lambda)$
$sk = (sk_{\mathsf{seed}}, sk_{\mathsf{HT}})$, $pk = pk_{\mathsf{HT}}$
Output $(pk, sk)$.

$\mathsf{SPHINCS.Sig}(sk, m)$
$z = \mathsf{PRF}_{\mathsf{msg}}(sk_{\mathsf{seed}}, m)$
$idx = \mathsf{PRF}_{\mathsf{idx}}(sk_{\mathsf{seed}}, m)$
$s_{idx} = \mathsf{PRF}_{\mathsf{seed}}(sk_{\mathsf{seed}}, idx)$
$(pk_{\mathsf{HORS}}, sk_{\mathsf{HORS}}) \leftarrow \mathsf{HORS.KeyGen}(1^\lambda; s_{idx})$
$\sigma_{\mathsf{HORS}} \leftarrow \mathsf{HORS.Sig}(sk_{\mathsf{HORS}}, z||m)$
$\sigma_{\mathsf{HT}} \leftarrow \mathsf{HT.Sig}(sk_{\mathsf{HT}}, pk_{\mathsf{FORS}}, idx)$
return $(idx, z, \sigma_{\mathsf{HT}}, \sigma_{\mathsf{FORS}})$.

$\mathsf{SPHINCS.Ver}(pk, m, (idx, z, \sigma_{\mathsf{HT}}, \sigma_{\mathsf{FORS}}))$
$pk_{\mathsf{HORS}} \leftarrow \mathsf{HORS.pkFromSig}(z||m, \sigma_{\mathsf{HORS}})$
return $\mathsf{HT.Ver}(pk_{\mathsf{HT}}, pk_{\mathsf{HORS}}, \sigma_{\mathsf{HT}}, idx)$
```

```
$\mathsf{SPHINCS+.KeyGen}(1^\lambda)$
$sk_{\mathsf{seed}} \leftarrow \{0,1\}^n$, $h_0 \leftarrow \mathcal{H}_0$
$(pk_{\mathsf{HT}}, sk_{\mathsf{HT}}) \leftarrow \mathsf{HT.KeyGen}(1^\lambda)$
$sk = (sk_{\mathsf{seed}}, sk_{\mathsf{HT}})$, $pk = (pk_{\mathsf{HT}}, h_0)$
Output $(pk, sk)$.

$\mathsf{SPHINCS+.Sig}(sk, m)$
$z = \mathsf{PRF}_{\mathsf{msg}}(sk_{\mathsf{seed}}, m)$
$idx = h_0(z||m)$
$s_{idx} = \mathsf{PRF}_{\mathsf{seed}}(sk_{\mathsf{seed}}, idx)$
$(pk_{\mathsf{FORS}}, sk_{\mathsf{FORS}}) \leftarrow \mathsf{sFORS.KeyGen}(1^\lambda; s_{idx})$
$\sigma_{\mathsf{FORS}} \leftarrow \mathsf{sFORS.Sig}(sk_{\mathsf{FORS}}, z||m)$
$\sigma_{\mathsf{HT}} \leftarrow \mathsf{HT.Sig}(sk_{\mathsf{HT}}, pk_{\mathsf{FORS}}, idx)$
return $(z, \sigma_{\mathsf{HT}}, \sigma_{\mathsf{FORS}})$.

$\mathsf{SPHINCS+.Ver}(pk, m, (z, \sigma_{\mathsf{HT}}, \sigma_{\mathsf{FORS}}))$
$idx = h_0(z||m)$
$pk_{\mathsf{FORS}} \leftarrow \mathsf{sFORS.pkFromSig}(z||m, \sigma_{\mathsf{FORS}})$
return $\mathsf{HT.Ver}(pk_{\mathsf{HT}}, pk_{\mathsf{FORS}}, \sigma_{\mathsf{HT}}, idx)$
```

Fig. 7. The framework of SPHINCS and SPHINCS+.

5 Quantum Chosen Message Attacks on SPHINCS(+)

In this section, we propose quantum chosen attacks on SPHINCS(+). The time complexity (quantified with the number of signing queries and hash computations required in the attack) is much lower than the optimal chosen message attacks.

5.1 Quantum Chosen Message Attacks on SPHINCS

Let q_s be the number of signing queries. The optimal attack requires approximately 2^{128} hash queries to break the EUF-CMA security of SPHINCS-256 when $q_s = 2^{50}$. Approximately the same number of hash queries are needed to break the EUF-CMA security of SPHINCS+-256s when $q_s = 2^{64}$. Our attack shows that if the signing oracle is quantum-accessible, the security will be lower.

In SPHINCS, $idx = \mathsf{PRF}_{\mathsf{idx}}(sk_{\mathsf{seed}}, \mathsf{PRF}_{\mathsf{msg}}(sk_{\mathsf{seed}}, m))$. Fix sk_{seed} and let $f(m) = \mathsf{PRF}_{\mathsf{idx}}(sk_{\mathsf{seed}}, \mathsf{PRF}_{\mathsf{msg}}(sk_{\mathsf{seed}}, m))$ be the function mapping m to idx. Since idx is a part of the signature, f can be computed by querying the signing oracle. By using Grover's algorithm, one can search a message m mapping to any index after $O(2^{h/2})$ queries to the signing oracle.

Note that any index authenticates a key pair of the few-time signature scheme. By repeating the above steps r times, one can obtain r message-signature pairs w.r.t. the same few-time signature key pair. If the secret key of the few-time signature is used too many times, the security level will be degraded rapidly.

The formal description of the attack on SPHINCS is as follows:

1. Denote $f(m) = \mathsf{PRF}_{\mathsf{idx}}(sk_{\mathsf{seed}}, \mathsf{PRF}_{\mathsf{msg}}(sk_{\mathsf{seed}}, m))$ be the function that maps the message m to the index $idx \in \{0,1\}^h$. Randomly pick $idx^* \in \{0,1\}^h$, and denote predicate $F(m) = 1$ iff $f(m) = idx^*$. Here, f and F are quantum-computable by querying the signing oracle $|\mathsf{SigO}\rangle$ since $|\mathsf{SigO}\rangle$ maps $|m, 0^h\rangle$ to $|m, f(m)\rangle$ by discarding the signature register except the index part.
2. Run Grover's algorithm on $F(m)$. It returns a random m such that $F(m) = 1$. It implies a HORS signature labeled by idx^*. This requires $O(2^{h/2})$ quantum queries to the signing oracle.[5]
3. Repeat the previous step r times. This requires $O(r2^{h/2})$ quantum queries to the signing oracle.[6] Let S be the set of labels of which the preimages have appeared in the HORS signatures. In other words, let $m^{(1)}, ..., m^{(r)}$ be the outputs of the Grover's algorithm and $z^{(j)}$ be the pseudorandomness z in the signature of $m^{(j)}$, then $S = \{h_i(z^{(j)}||m^{(j)})\}_{i\in[k],j\in[r]}$.

   Note that for each s_j, the probability of appearing in a HORS signature is k/t. The probability of appearing in r random signatures is $1-(1-k/t)^r$. Thus, the expectation of $|S|$ is
$$\mathbb{E}[|S|] = \left(1 - \left(1 - \frac{k}{t}\right)^r\right) \cdot t \geq (1 - e^{-\frac{kr}{t}}) \cdot t,$$
   where the inequality comes from $(1-x)^\alpha \leq e^{-\alpha x}$.
4. Denote function $G(z||m) = 1$ iff $\{h_i(z||m)\}_{i\in[k]} \subset S$. Run Grover's algorithm on G. It outputs $z^*||m^*$ whose corresponding preimages have appeared in the HORS signatures. The expected number of quantum hash queries in this step is
$$O\left(\sqrt{\left(\frac{t}{|S|}\right)^k}\right) = O((1 - e^{\frac{kr}{t}})^{-\frac{k}{2}}).$$
5. Note that the signing oracle has been queried $q_s = O(r2^{h/2})$ times. The attacker needs to return at least (q_s+1) forgeries. Step 4 needs to be repeated $(q_s + 1 - r)$ times. (It is unnecessary to compute σ_{HT} for the new forgeries since all the forgeries share a common σ_{HT}.) The total number of hash queries is
$$q_H = O(q_s + 1 - r) \cdot O((1 - e^{-\frac{kr}{t}})^{-\frac{k}{2}}) = O((1 - e^{-\frac{kr}{t}})^{-\frac{k}{2}} \cdot r2^{\frac{h}{2}}).$$

[5] In this paper, we always use $O(\cdot)$ to describe the (upper-bound) number of queries due to the implement of Grover's algorithm. Note that Grover's search could fail in the case that the number of preimages is unknown (we only know the expected number). It can be solved by repeating it constant times [25]. Since the repetition only causes (at most) a constant factor on the time complexity, we ignore its effect and only focus on the complexity related to the parameters of SPHINCS(+) and r.

[6] We require the r results be distinct. It can be guaranteed by a simple modification. Let $i \in [r]$ and $f^{(i)}(m) = f(C_i||m)$, where C_i denotes the binary expression of i. Define $F^{(i)}(m) = 1$ iff $f^{(i)}(m) = idx^*$ and run Grover's algorithm on $F^{(i)}$ for each $i \in [r]$ instead. Thus, the results have distinct prefix of length $\lceil \log r \rceil$.

In SPHINCS-256, $k = 32$, $t = 2^{16}$ and $h = 60$. When $r = 2^{10}$, q_s and q_H are approximately 2^{40} and 2^{61}, respectively. When $r = 2^{14}$, q_s reaches 2^{43} and q_H decreases to 2^{43} as well. It is much lower than the level of EUF-CMA security, where $q_s = 2^{50}$, and q_H is expected to be 2^{128}.

5.2 Quantum Chosen Message Attacks on SPHINCS+

In SPHINCS+, the index is not directly contained in the signature. Instead, the index is computed by $idx = h_0(z||m)$, where z is a (pseudo-)randomizer and part of the signature. It is not a big issue since we can modify the condition on Grover's algorithm to find a malicious randomizer z mapping to our malicious index. In this section, we simply denote by H the hash computation of h_0 and $(h_1, ..., h_k)$. (In practice, h_0 and $(h_1, ..., h_k)$ are parts of a function H.)

Our attack on SPHINCS+ is as follows:

1. Let $z(m)$ be the function mapping from m to the corresponding z (which can be evaluated by a signing query). For some $idx^* \in \{0,1\}^h$, denote predicate $F(m) = 1$ iff $h_0(z(m)||m) = idx^*$. Similarly, F is quantum-computable by querying $|\mathsf{SigO}\rangle$ and a quantum query to h_0.
2. Run Grover's algorithm on $F(m)$. It outputs a random m such that $F(m) = 1$. It implies a sFORS signature labeled by idx^*. This requires $O(2^{h/2})$ quantum queries to the signing oracle and $O(2^{h/2})$ quantum queries to h_0.
3. Repeat the previous step r times. This requires $O(r2^{h/2})$ quantum queries to the signing oracle. For $i \in [k]$, let S_i be the set of labels of which the preimages have appeared in the i-th tree of sFORS signatures. In other words, $S_i = \{h_i(z(m^{(j)})||m^{(j)})\}_{j\in[r]}$.

   For each $s_{i,j}$, the probability of appearing in an sFORS signature is $1/t$. The probability of appearing in r random signatures is $1-(1-1/t)^r$. Thus, the expectation of $|S_i|$ is

$$\mathbb{E}[|S_i|] = \left(1 - \left(1 - \frac{1}{t}\right)^r\right) \cdot t \geq (1 - e^{-\frac{r}{t}}) \cdot t,$$

   and thus

$$\mathbb{E}\left[\prod_{i\in[k]} |S_i|\right] \geq (1 - e^{-\frac{r}{t}})^k \cdot t^k.$$

4. Denote function $G(z||m) = 1$ iff $\forall i \in [k], h_i(z||m) \in S_i \wedge h_0(z||m) = idx^*$. Run Grover's algorithm on G. It outputs $z^*||m^*$ whose corresponding preimages have appeared in sFORS signatures. The expected number of quantum hash queries in this step is

$$O\left(\sqrt{\left(\frac{2^h \cdot t^k}{\prod_{i\in[k]} |S_i|}\right)}\right) = O((1 - e^{-\frac{r}{t}})^{-\frac{k}{2}} \cdot 2^{\frac{h}{2}}).$$

5. The signing oracle has been queried $q_s = O(r2^{h/2})$ times. The forgery needs to contain at least $(q_s + 1)$ forgeries. Step 4 needs to be repeated $(q_s + 1 - r)$ times. The total number of hash queries is

$$q_H = O(r2^{\frac{h}{2}}) + O(q_s + 1 - r) \cdot O((1 - e^{-\frac{r}{t}})^{-\frac{k}{2}} \cdot 2^{\frac{h}{2}}) = O((1 - e^{-\frac{r}{t}})^{-\frac{k}{2}} \cdot r2^h).$$

In SPHINCS+-256s, $k = 22$, $t = 2^{14}$ and $h = 64$. When $r = 2^{16}$, q_s and q_H are approximately 2^{48} and 2^{80}, respectively. It is also lower than the level of EUF-CMA security, where $q_s = 2^{64}$ and q_H is expected to be 2^{128}.

Remark 4. In a quantum algorithm, it may be problematic to maintain a quantum state with large quantum memory (especially for HBS schemes due to the large size of signatures). However, it is not an big issue since our attacks only need the index *idx* (or the randomizer z) in the iterations of Grover's algorithm, and main parts of the quantum responses can be discarded.

5.3 Attacks in the BU Model

Note that in the above attacks, the adversary has had the ability to forge a signature for any message before the final step. Thus, if we use the BU model instead, the adversary only need to forge one signature for a message in $B_{\varepsilon,n}$, and the large number of computations in the final step can be saved.

In the following, we omit the parameter n and use B_ϵ to denote $B_{\epsilon,n}$. The strategy of attacking SPHINCS in the BU model is as follows.

1. Denote $f(m) = \mathsf{PRF}_{\mathsf{idx}}(sk_{\mathsf{seed}}, \mathsf{PRF}_{\mathsf{msg}}(sk_{\mathsf{seed}}, m))$ be the function that maps the message m to the index $idx \in \{0,1\}^h$. For some $idx^* \in \{0,1\}^h$, denote predicate $F(m) = 1$ iff $f(m) = idx^*$ and $m \notin B_\varepsilon$. Here, F is quantum-computable by querying the signing oracle $B_\varepsilon\mathsf{SigO}$.
2. Run Grover's algorithm on $F(m)$. This requires $O\Big(\sqrt{\frac{2^h}{1-\varepsilon}}\Big)$ queries to $B_\varepsilon\mathsf{SigO}$.
3. Repeat the last step r times and denote S as above. The expectation of $|S|$ is also $(1 - e^{-\frac{kr}{t}}) \cdot t$.
4. Denote predicate $F'(m) = 1$ iff $B_\varepsilon\mathsf{SigO}(m) = \perp$. It outputs a message $m^* \in B_\varepsilon$. This requires approximately $O(\varepsilon^{-1/2})$ signing queries.
5. Denote function $G(z) = 1$ iff $\{h_i(z||m^*)\}_{i\in[k]} \subset S$. Run Grover's algorithm on G. It outputs z^* such that the preimages corresponding to (z^*, m^*) have appeared in S. The expected number of quantum hash queries in this step is also $O((1 - e^{\frac{kr}{t}})^{-\frac{k}{2}})$.
   Then the adversary successfully generates a forgery $\sigma^* = (idx^*, z^*, \sigma^*_{\mathsf{HT}}, \sigma^*_{\mathsf{HORS}})$ for $m^* \in B_\varepsilon$, where σ^*_{HT} is obtained by an additional signing query and σ^*_{HORS} is obtained by S.

The total number of required (quantum) queries is respectively

$$q_s = O(r2^{\frac{h}{2}}(1 - \varepsilon)^{-\frac{1}{2}}) + O(\varepsilon^{-\frac{1}{2}}), \quad q_H = O((1 - e^{-\frac{kr}{t}})^{-\frac{k}{2}}).$$

Note that q_s is slightly larger than the original attack (increased by a polynomial $\sqrt{\varepsilon}$) and q_H decreases to $1/r2^{h/2}$ of the original one.

The above attack also works on SPHINCS+, but we have another one that requires less queries. The main idea is similar. First, we find a message m^* in the blind region. Then, to sign a message for m^*, we need an sFORS signature associated with some index idx^*. Note that the sFORS signature includes k elements. We directly use Grover's algorithm to search k messages (outside of the blind region) that respectively map to the k target elements. It can be done by quantum signing queries. Finally, the sFORS signature of m^* is covered by the k signatures, and a forgery is generated. The attack is as follows.

1. Find m^* such that $B_\varepsilon \mathsf{SigO}(m^*) = \perp$. This requires $O(\varepsilon^{-\frac{1}{2}})$ quantum hash queries to $B_\varepsilon \mathsf{SigO}$.
2. Pick $z^* \in \{0,1\}^n$ and let $idx^* = h_0(z^*||m^*)$. Let function $z(m)$ be the map from m to the corresponding z. For $i \in [k]$, denote predicate $F_i(m) = 1$ iff (1) $z(m) \neq \perp$, (2) $h_0(z(m)||m) = idx^*$, and (3) $h_i(z(m)||m) = h_i(z^*||m^*)$. F_i is quantum-computable by querying $B_\varepsilon \mathsf{SigO}$ and a quantum query to h_0, h_i. Run Grover's algorithm on F_i. The expected number of F_i computations is $O(\sqrt{(1-\varepsilon)^{-1} \cdot 2^h \cdot t})$.
3. After that, the secret values in sFORS signature on m^* is covered the k signatures. A forgery is then generated.

In total, the number of required (quantum) queries is respectively

$$q_s = O(k2^{\frac{h+\log t}{2}}(1-\varepsilon)^{-\frac{1}{2}}) + O(\varepsilon^{-\frac{1}{2}}), \quad q_H = O(k2^{\frac{h+\log t}{2}}(1-\varepsilon)^{-\frac{1}{2}}).$$

With the parameters in SPHINCS+-256s, q_s and q_H are both approximately 2^{43}, which is lower than our attack in the PO model.

The concrete complexity of our attacks is summarized in Table 1.

6 SPHINCS-FORS: A Provably Secure HBS Scheme Against Quantum Chosen Message Attacks

In this section, we propose a simple variant of SPHINCS+ with provable security in the sense of EUF-qCMA.

6.1 Generic Security of Few-Time HBS Schemes

As a preparatory work, we begin with the security of the few-time HBS schemes, which are used as building blocks in SPHINCS(+). To the best of our knowledge, no quantum generic security bound is given before this work (even in the sense of EUF-CMA). We fill the gap with the following theorems.

Theorem 3. *Let the hash functions in HORS and sFORS be modeled as quantum random oracles. It holds that*

$$Adv^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HORS},r,q}(\mathcal{A}) \leq O\left(q^{2(r+1)}\left(\frac{rk}{t}\right)^k + \frac{q^2kt}{2^n}\right),$$

$$Adv^{\textsf{EUF-CMA}}_{\textsf{sFORS},r,q}(\mathcal{A}) \leq O\left(q^{2(r+1)}\left(\frac{r}{t}\right)^k + \frac{q^2}{2^n}\right),$$

$$Adv^{\textsf{EUF-qCMA}}_{\textsf{sFORS},r,q}(\mathcal{A}) \leq O\left(q^{2(r+1)}\left(\frac{r^2}{t}\right)^k + \frac{q^2 k t^r}{2^n}\right).$$

Since the EUF-CMA security of HORS (and FORS) is reduced to (restricted) subset resilience and (variants of) preimage resistance [31,33], the EUF-CMA security bounds are directly implied from Theorem 1.

We show the proof sketch of the EUF-qCMA security in particular. We reduce the EUF-qCMA security to weak subset resilience and (variants) of preimage resistance, following the idea of the security proof of Lamport's scheme in [11]. Assume $\mathcal{H}$ is weak subset-resilient and the adversary eventually outputs forgeries for (distinct) $m_1, ..., m_{r+1}$. There must exists some $i^* \in [k]$ such that $h_{i^*}(m_1), ..., h_{i^*}(m_{r+1})$ are distinct. As a hybird argument, we measure the values of h_{i^*} in the signing queries[7]. After the measurement, the signing oracle will only return *one* of the preimages at position i^* in each signing query, but the adversary is required to output one more of them, which can be used to break preimage resistance. From Lemma 2, each partial measurement only causes security loss of t, a polynomial number. Since the partial measurements are performed r times, a constant number, the overall security loss is still polynomial. See details in the full version [34].

Theorem 4. *Let the hash functions in rFORS be modeled as quantum random oracles. It holds that*

$$Adv^{\textsf{EUF-CMA}}_{\textsf{rFORS},r,q}(\mathcal{A}) \leq O\left(\sqrt{\frac{q}{2^n}} + q^2\left(\frac{r}{t}\right)^k + \frac{q^2}{2^n}\right),$$

$$Adv^{\textsf{EUF-qCMA}}_{\textsf{rFORS},r,q}(\mathcal{A}) \leq O\left(q^{2(r+1)}\left(\frac{r^2}{t}\right)^k + \frac{q^2 k t^r}{2^n}\right).$$

The EUF-CMA security can be simply reduced to target subset resilience, while the EUF-qCMA security is immediately implied from Theorem 3. See details in the full version [34].

6.2 Discussion: How to Avoid Quantum Attacks?

We then discuss how to construct a many-time signature scheme with provable security equipped with above few-time HBS schemes.

In the attacks in Sect. 5, the key idea is to search messages that map to a single index idx^*. The search is done by iteratively running a function F in Grover's algorithm. A simple improvement to avoid these attacks is making F randomized. That is, in each signing operation, the signer adds a random nonce

[7] We cannot learn from the final forgeries about which i^* should be targeted, since the measurements should be performed on-line. Instead, we guess it from $[k]$ initially.

in calculating the pseudorandomness $z = \mathsf{PRF}_{\mathsf{msg}}(sk_{\mathsf{seed}}, m)$. It is indeed the probabilistic version of SPHINCS+ [8]. Note that the nonce does not affect the security reduction of EUF-CMA, but does affect the EUF-qCMA security.

Unfortunately, we cannot give a *security proof* of EUF-qCMA security for the above variants, even if the random nonce is well sampled. Note that the security under quantum chosen message attacks of SPHINCS(+) is more complicated for the following reasons. First, in the classical setting, a response of the signing query contains only one few-time signature. Since *idx* may differ in superpositions in the quantum-access setting, a quantum SPHINCS(+) signature may contain many few-time signatures for *multiple* key pairs. This multi-instance case exceeds the discussion in Sect. 6.1. Second, a quantum SPHINCS(+) signature may also contain a large number of HT signatures σ_{HT} in superpositions. It makes the analysis even more complicated.

So how do we construct a *provably secure* hash-based signature scheme under qCMAs? Our solution is simple. The first step is to make each signing response only contain few-time signatures related to *one* key pair. For this purpose, we make the index of the few-time signature independent to the message. In each signing query, the signing algorithm randomly picks a leaf from $\{0,1\}^h$ instead of running the pseudorandom function on the message. Since the randomness of a signing query is global, the resulting signatures in superpositions share common randomness and thus a common *idx*, implying a common few-time signature key pair. In addition, note that σ_{HT} is the signature on the few-time signature public key. Since all superpositions share a common public key, the resulting σ_{HT} is also identical in all superpositions. The security is then reduced to the quantum-access security of the few-time signature scheme in the single-instance case, which has been evaluated in Sect. 6.1.

This variant can avoid the above attacks since the index is independent of the message. However, note that the random index needs to be directly contained in the signature, so an adversary can arbitrarily choose an index in the forgeries. It causes lower security than SPHINCS+, especially in the EUF-CMA model. Thus, the classical security also needs re-analyzed.

In the next subsection, we present SPHINCS-FORS, a variant of SPHINCS+ that follows the approach and provides provable EUF-qCMA security.

6.3 SPHINCS-FORS

Construction 1. *Let* $\mathsf{PRF}_{\mathsf{seed}} : \{0,1\}^n \times \{0,1\}^h \to \{0,1\}^n$ *be a pseudorandom function, and* rFORS *and* HT *be depicted in the full version [34]. SPHINCS-FORS is depicted in Fig. 8.*

The difference from SPHINCS+ is as follows:

- The strategies for choosing the index are different. In SPHINCS-FORS, the index is truly random in $\{0,1\}^h$ while in SPHINCS+ it is pseudorandom related to m. The random index is directly contained in the resulting signature.

```
SPHINCS-FORS.KeyGen($1^\lambda$)
    $sk_{\mathsf{seed}} \leftarrow \{0,1\}^n$,
    $(pk_{\mathsf{HT}}, sk_{\mathsf{HT}}) \leftarrow \mathsf{HT.KeyGen}(1^\lambda)$
    $sk = (sk_{\mathsf{seed}}, sk_{\mathsf{HT}})$, $pk = pk_{\mathsf{HT}}$
    Output $(pk, sk)$.

SPHINCS-FORS.Sig($sk, m$)
    $idx \leftarrow \{0,1\}^h$
    $s_{idx} = \mathsf{PRF}_{\mathsf{seed}}(sk_{\mathsf{seed}}, idx)$
    $(pk_{\mathsf{FORS}}, sk_{\mathsf{FORS}}) \leftarrow \mathsf{rFORS.KeyGen}(1^\lambda; s_{idx})$
    $(z, \sigma_{\mathsf{FORS}}) \leftarrow \mathsf{rFORS.Sig}(sk_{\mathsf{FORS}}, m)$
    $\sigma_{\mathsf{HT}} \leftarrow \mathsf{HT.Sig}(sk_{\mathsf{HT}}, pk_{\mathsf{FORS}}, idx)$
    return $(idx, z, \sigma_{\mathsf{HT}}, \sigma_{\mathsf{FORS}})$.

SPHINCS-FORS.Ver($pk, m, (idx, z, \sigma_{\mathsf{HT}}, \sigma_{\mathsf{FORS}})$)
    $pk_{\mathsf{FORS}} \leftarrow \mathsf{rFORS.pkFromSig}(m, (z, \sigma_{\mathsf{FORS}}))$
    return $\mathsf{HT.Ver}(pk_{\mathsf{HT}}, pk_{\mathsf{FORS}}, \sigma_{\mathsf{HT}}, idx)$
```

Fig. 8. The framework of SPHINCS-FORS

- In SPHINCS-FORS, we use rFORS as the few-time signature. A minor difference is that in SPHINCS+, the (pseudo-)randomizer z is calculated by a global function $\mathsf{PRF}_{\mathsf{msg}}(\mathsf{sk}_{\mathsf{seed}}, m)$ while in SPHINCS-FORS, sk_{seed} differs in different indices.

6.4 Security Analysis

In this subsection, we analyze the security of SPHINCS-FORS under (quantum) chosen message attacks.

At first, we need to evaluate the security for HT. As we use the same HT as SPHINCS+ and its (EUF-sNACMA) security has been evaluated in [8,25], we omit the formal analysis of HT. The success probability of breaking the security is at most $O(q_H \cdot 2^{-n/2})$, where q_H denotes the number of hash queries.

For a signature scheme Γ, let $\mathrm{InSec}^{*}_{\Gamma,r,q_H}(\xi)$ be the maximum of $\mathrm{Adv}^{*}_{\Gamma,r,q_H}(\mathcal{A})$ for all ξ-time adversary $\mathcal{A}$ and $* \in \{\mathsf{EUF\text{-}CMA}, \mathsf{EUF\text{-}qCMA}, \mathsf{EUF\text{-}sNACMA}\}$. The security of SPHINCS-FORS is proven as follows.

Theorem 5. *For any ξ-time adversary $\mathcal{A}$, it holds that*

$$\begin{aligned} Adv^{*}_{\textsf{SPHINCS-FORS},q_s,q_H}(\mathcal{A}) \leq & InSec^{\textsf{EUF-sNACMA}}_{\textsf{HT},2^h,q_H}(\xi) + InSec^{\textsf{Ind-PRF}}_{\textsf{PRF}_{\textsf{seed}},2^h}(\xi) \\ & + \sum_{r=0}^{q_s} p(r, q_s) \cdot InSec^{*}_{\textsf{rFORS},r,q_H}(\xi), \end{aligned}$$

where $p(r, q_s) = \min\{2^{r(\log q_s - h) + h - \log r!}, 2^h\}$ *and* $* \in \{\textsf{EUF-CMA}, \textsf{EUF-qCMA}\}$.

Proof. We only show the proof for EUF-qCMA security here. The proof for EUF-CMA security is very similar and thus omitted.

As mentioned in Sect. 6.2, all the signatures contained in a response from the signing oracle share a common index and thus a common rFORS key pair. In each superposition, idx and σ_{HT} are identical. The only "quantum part' of a response is $(z, \sigma_{\mathsf{FORS}})$, the signature of rFORS. It implies that the EUF-qCMA security of SPHINCS-FORS is reduced to the EUF-qCMA security of rFORS and the classical security of HT.

The statement can be proven by the following hybrid arguments.

- **Game 0** is the original EUF-qCMA experiment of SPHINCS-FORS.
- **Game 1** differs from **Game 0** in that, in the signing oracle, s_{idx} is calculated by $\mathsf{TRF}(idx)$ where $\mathsf{TRF} : \{0,1\}^h \rightarrow \{0,1\}^\lambda$ is a truly random function. If the success probability differs, it implies a reduction distinguishing $\mathsf{PRF}_{\mathsf{seed}}$ and TRF. Note that there are at most 2^h calls to PRF. We have

$$|\Pr[\textbf{Game 1}] - \Pr[\textbf{Game 0}]| \leq \mathrm{InSec}^{\mathsf{Ind\text{-}PRF}}_{\mathsf{PRF}_{\mathsf{seed}},2^h}(\xi).$$

- **Game 2** differs from **Game 1** as follows. After $\mathcal{A}$ outputs (q_s+1) message-signature pairs, check whether there exists a forgery $(m^*, \Sigma^*) = (m^*, (idx^*, z, \sigma^*_{\mathsf{FORS}}, \sigma^*_{\mathsf{HT}}))$ such that $pk^*_{\mathsf{FORS}} \neq pk_{\mathsf{FORS}}$, where $pk^*_{\mathsf{FORS}} \leftarrow \mathsf{rFORS.pkFromSig}(m^*, (z, \sigma^*_{\mathsf{FORS}}))$, $s_{idx^*} = \mathsf{TRF}(idx^*)$ and $pk_{\mathsf{FORS}} \leftarrow \mathsf{rFORS.KeyGen}(1^n; s_{idx^*})$. If so, it returns 0.

  Game 2 differs from **Game 1** only if the adversary generates a HT signature for a "fake" pk^*_{FORS} which is not consistent to the real one. It implies a reduction attacking the EUF-sNACMA security of HT. At the beginning, the reduction generates the rFORS public keys w.r.t. all the indices in $\{0,1\}^h$ and sends them with the corresponding indices to the challenger. Then, it obtains the HT signatures and the public key pk_{HT} from the challenger. When signing a message m from the adversary, it picks a random $idx \in \{0,1\}^h$, generates the corresponding rFORS signature $(z, \sigma_{\mathsf{FORS}})$. Then, it replies with $(idx, z, \sigma_{\mathsf{FORS}})$ and the corresponding σ_{HT} from the challenger. Finally, the adversary outputs a pk^*_{FORS} that is different from the real one. It implies a forgery of HT with state idx^*. We have

$$\Pr[\textbf{Game 1}] \leq \Pr[\textbf{Game 2}] + Adv^{\mathsf{EUF}}_{\mathsf{HT},2^h,q_H}\text{-}\mathsf{sNACMA}(\mathcal{A}).$$

  In **Game 2**, the adversary wins only if it generates (q_s+1) rFORS forgeries (of multiple instances) after q_s signing queries. Note that in each signing query, the signing oracle picks one $idx \in \{0,1\}^h$ and signs the message by the rFORS key pair associated with idx. Due to the pigeonhole principle, there must exist a special $idx^* \in \{0,1\}^h$ that has been used r times in signing queries and is used at least $(r+1)$ times in the forgeries for some $r \geq 0$.
- **Game 3** differs from **Game 2** in that it guesses $idx' \in \{0,1\}^h$ at the beginning of the experiment and outputs 0 if $idx' \neq idx^*$. We have

$$\Pr[\textbf{Game 2}] \leq 2^h \cdot \Pr[\textbf{Game 3}].$$

- In **Game 3**, $\mathcal{A}$ wins if it generates $(r+1)$ forgeries for the rFORS key pair associated with idx' conditioned that it is chosen r times in q_s signing queries. It breaks the EUF-qCMA security of rFORS with r signing queries. In addition, let $\mathbf{E}_r$ be the event that a leaf is chosen r times. The probability of $\mathbf{E}_r$ is $\binom{q_s}{r}(1-2^{-h})^{q_s-r}(2^{-h})^r < 2^{r(\log q_s - h) - \log r!}$. Thus, we have

$$\begin{aligned}\Pr[\mathbf{Game\ 3}] &= \sum_{r=0}^{q_s} \Pr[\mathbf{Game\ 3}|\mathbf{E}_r] \cdot \Pr[\mathbf{E}_r] \\ &\leq \sum_{r=0}^{q_s} \mathrm{InSec}^{\textsf{EUF}}_{FORS,r,q_H}\text{-}\textsf{qCMA}(\xi) \cdot \min\{2^{r(\log q_s - h) - \log r!}, 1\}.\end{aligned}$$

This completes the proof.

From Theorem 4 and 5, we have

Corollary 3. *Let the hash functions in SPHINCS-FORS be modeled as quantum random oracles. It holds that*

$$Adv^{\textsf{\textit{EUF-qCMA}}}_{\textsf{\textit{SPHINCS-FORS}},q_s,q_H}(\mathcal{A}) \leq O\left(\frac{q_H}{2^{n/2}} + \sum_{r=0}^{q_s} p(r,q_s) \cdot \min\left\{q_H^{2(r+1)}\left(\frac{r^2}{t}\right)^k + \frac{q_H^2 k t^r}{2^n}, 1\right\}\right), \tag{8}$$

$$Adv^{\textsf{\textit{EUF-CMA}}}_{\textsf{\textit{SPHINCS-FORS}},q_s,q_H}(\mathcal{A}) \leq O\left(q_s\sqrt{\frac{q_H+q_s}{2^n}} + \frac{q_H}{2^{n/2}} + q_H^2 \sum_{r=0}^{q_s} p(r,q_s)\left(\frac{r}{t}\right)^k\right). \tag{9}$$

Note that here the term caused by adaptive reprogramming is $O\left(q_s\sqrt{\frac{q_H+q_s}{2^n}}\right)$ rather than $\sum_{r=0}^{q_s} p(r,q_s) \cdot \frac{3r}{2}\sqrt{\frac{q_H+r+1}{2^n}}$. It is because that the random oracle is reprogrammed at most q_s times in the reduction. For the same reason the terms caused by SM-TCR, SM-DSPR and Ind-PRF are gathered to $O(q_H \cdot 2^{-n/2})$.

6.5 Concrete Security

As a variant of SPHINCS+, SPHINCS-FORS is expected to reach (at least) the same security level as SPHINCS+. However, we note that the EUF-CMA security level of SPHINCS-FORS is lower than SPHINCS+ with the same parameters. It is because of the different strategies of choosing the index. Since the index is directly contained in the signature (just like what SPHINCS does), the adversary is able to arbitrarily choose a target index to forge a signature. In the full version [34], we show a concrete attack on SPHINCS-FORS, implying the difference in security levels with SPHINCS+.

Thus, we need to increase some parameters to reach the same level as SPHINCS+ in the sense of EUF-CMA, and meanwhile provide provable security in the sense of EUF-qCMA. As a result, the signature size and running time will become larger.

We give two instances with different parameters and security levels. See details of the parameters in Table 2.

- As is observed in Sect. 5.2, (deterministic) SPHINCS+-256s can provide *at most* 80-bit quantum-access security when $q_s = 2^{48}$. By adapting the parameters k, t, and h, we result in SPHINCS-FORS v1 that can provide *at least* 80-bit security in the sense of provable quantum-access security.
- If we directly increase h in SPHINCS+-256s to 104, say SPHINCS+-256s$*$, it can provide *at most* 128-bit quantum-access security (due to our attacks) when $q_s = 2^{64}$. On the other hand, by adapting the parameters, we result in SPHINCS-FORS v2 that can provide *at least* 128-bit security.

7 Conclusion and Open Questions

This paper analyzes the quantum-access security of HBS schemes in two directions. First, we show quantum chosen message attacks (or superposition attacks) on stateless HBS schemes, such as SPHINCS and SPHINCS+. The time complexity of the quantum chosen message attacks is lower than the optimal attacks in the classical setting. Next, we propose a variant of SPHINCS+ called SPHINCS-FORS. It is a provably secure HBS scheme against quantum chosen message attacks. As far as we know, it is the first practical HBS scheme with provable security against quantum chosen message attacks.

Note that our attacks do not work on the probabilistic version of SPHINCS+. Since there is no security proof, it is an open question whether probabilistic SPHINCS+ is secure against quantum chosen message attacks. In addition, our security bound of SPHINCS-FORS against quantum chosen message attacks is possibly non-tight. It shows a lower bound of the security, but we are not aware of any concrete attacks that reach this bound. It is also an open question whether we can get a tighter bound or if there exists a better attack.

Acknowledgement. We would like to thank Tsuyoshi Takagi for valuable suggestions and anonymous reviewers of ACISP 2023 for helpful comments. Quan Yuan is supported by JST CREST Grant Number JPMJCR2113, Japan.

References

1. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. J. ACM **51**(4), 595–605 (2004)
2. Alagic, G., Majenz, C., Russell, A., Song, F.: Quantum-access-secure message authentication via blind-unforgeability. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 788–817. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_27
3. Aumasson, J.-P., et al.: SPHINCS+ - submission to the NIST post-quantum project, vol. 3 (2020)
4. Aumasson, J.-P., Endignoux, G.: Clarifying the subset-resilience problem. Cryptology ePrint Archive, Report 2017/909 (2017)
5. Aumasson, J.-P., Endignoux, G.: Improving stateless hash-based signatures. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 219–242. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76953-0_12

6. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, pp. 62–73 (1993)
7. Bernstein, D.J., et al.: SPHINCS: practical stateless hash-based signatures. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 368–397. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_15
8. 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, pp. 2129–2146. Association for Computing Machinery (2019)
9. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3
10. Boneh, D., Zhandry, M.: Quantum-secure message authentication codes. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 592–608. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_35
11. Boneh, D., Zhandry, M.: Secure signatures and chosen ciphertext security in a quantum computing world. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 361–379. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_21
12. Bonnetain, X., Hosoyamada, A., Naya-Plasencia, M., Sasaki, Yu., Schrottenloher, A.: Quantum attacks without superposition queries: the offline Simon's algorithm. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 552–583. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_20
13. Bouaziz-Ermann, S., Grilo, A.B., Vergnaud, D.: Quantum security of subset cover problems. Cryptology ePrint Archive: Report 2022/1714 (2022)
14. Buchmann, J., Dahmen, E., Hülsing, A.: XMSS - a practical forward secure signature scheme based on minimal security assumptions. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 117–129. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_8
15. Chatterjee, R., Chung, K.M., Liang, X., Malavolta, G.: A note on the post-quantum security of (ring) signatures. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography (PKC 2022). LNCS, vol. 13178, pp. 407–436. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97131-1_14
16. Cremers, C., Düzlü, S., Fiedler, R., Fischlin, M., Janson, C.: Buffing signature schemes beyond unforgeability and the case of post-quantum signatures. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 1696–1714. IEEE (2021)
17. Gagliardoni, T., Hülsing, A., Schaffner, C.: Semantic security and indistinguishability in the quantum world. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 60–89. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_3
18. Garg, S., Yuen, H., Zhandry, M.: New security notions and feasibility results for authentication of quantum data. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 342–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_12
19. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Campbridge University Press, Cambridge, UK (2004)
20. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)

21. Grilo, A.B., Hövelmanns, K., Hülsing, A., Majenz, C.: Tight adaptive reprogramming in the QROM. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13090, pp. 637–667. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92062-3_22
22. 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)
23. Hosoyamada, A., Iwata, T.: 4-round Luby-Rackoff construction is a qPRP. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 145–174. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_6
24. Hosoyamada, A., Sasaki, Yu.: Cryptanalysis against symmetric-key schemes with online classical queries and offline quantum computations. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 198–218. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76953-0_11
25. Hülsing, A., Kudinov, M.: Recovering the tight security proof of SPHINCS$^+$. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology (ASIACRYPT 2022). LNCS, vol. 13794, pp. 3–33. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22972-5_1
26. Hülsing, A., Rijneveld, J., Schwabe, P.: ARMed SPHINCS. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 446–470. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_17
27. Hülsing, A., Rijneveld, J., Song, F.: Mitigating multi-target attacks in hash-based signatures. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 387–416. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_15
28. Lamport, L.: Constructing digital signatures from a one way function. Technical report, Technical Report SRI-CSL-98, SRI International Computer Science Laboratory (1979)
29. Majenz, C., Manfouo, C.M., Ozols, M.: Quantum-access security of the Winternitz one-time signature scheme. In: 2nd Conference on Information-Theoretic Cryptography (ITC 2021), vol. 199, pp. 21:1–21:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
30. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_21
31. Reyzin, L., Reyzin, N.: Better than BiBa: short one-time signatures with fast signing and verifying. In: Batten, L., Seberry, J. (eds.) ACISP 2002. LNCS, vol. 2384, pp. 144–153. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45450-0_11
32. Yamakawa, T., Zhandry, M.: Classical vs quantum random oracles. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 568–597. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_20
33. Yuan, Q., Tibouchi, M., Abe, M.: On subset-resilient hash function families. Des. Codes Cryptogr. **90**, 719–758 (2022). https://doi.org/10.1007/s10623-022-01008-4
34. Yuan, Q., Tibouchi, M., Abe, M.: Quantum-access security of hash-based signature schemes. Cryptology ePrint Archive, Report 2023/556 (2022)
35. Zhandry, M.: How to construct quantum random functions. In: 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, pp. 679–687 (2012)
36. Zhandry, M.: A note on the quantum collision and set equality problems. Quantum Inf. Comput. **15**(7–8), 557–567 (2015)

37. Zhandry, M.: How to record quantum queries, and applications to quantum indifferentiability. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 239–268. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_9
38. Zhang, K., Cui, H., Yu, Y.: SPHINCS-α: a compact stateless hash-based signature scheme. Cryptology ePrint Archive: Report 2022/59 (2022)

Tightly Secure Lattice Identity-Based Signature in the Quantum Random Oracle Model

Ernest Foo and Qinyi Li(✉)

Griffith University, Brisbane, Australia
qinyi.li@griffith.edu.au

Abstract. We present a quantumly secure identity-based signature scheme based on the standard short integer solution problem, featuring tight security reductions in the quantum and classic random oracle models. The scheme has short signatures. Each signature contains a single lattice vector plus a single bit. Compared to the existing tightly secure, short lattice identity-based signature schemes by Pan and Wagner (PQCrypto'21), our scheme has a shorter signature size (around 30% shorter), stronger unforgeability, relies on a weaker assumption, and has detailed proof in the quantum random oracle model.

Keywords: Identity-based signature · strong unforgeability · lattice · quantum random oracle model

1 Introduction

Designing cryptographic schemes often follows a reductionist approach. More specifically, a cryptographic scheme is constructed along with a security theorem, stating that in some pre-defined security model, a t-time adversary who breaks the cryptographic scheme with probability ϵ can be turned into a t'-time algorithm that solves some computational problem with probability $\epsilon' \geq \epsilon/\theta$ with $\theta \geq 1$ and $t \approx t'$. If the problem is believed hard, ϵ' would be very small (or negligible in the security parameter in the asymptotic sense), making ϵ small too. As a result, the cryptographic scheme is unlikely broken. The factor θ measures the *tightness* of the reductionist proof. We say a reduction is tight if θ is a small *constant* and a cryptographic scheme with a tight security reduction is tightly secure. Tight security reduction is an attractive feature, with which, the cryptographic scheme can be implemented with smaller parameters (e.g., key sizes) to reach the same security level ϵ, leading to better efficiency.

Many cryptographic schemes have tight reductions. Among them, only a few tightly secure cryptographic schemes rely on quantum-immune assumptions, e.g., computational assumptions from high-dimensional lattices [2–4,6,15,16]. Following the recent works of Pan and Wagner, [15,16], we focus on quantumly secure identity-based signature (IBS) schemes with short signatures and tight

Authors are ordered alphabetically.

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 381–402, 2023.
https://doi.org/10.1007/978-3-031-35486-1_17

security reductions from (believed) quantumly intractable lattice problems. Here "short signature" means a signature contains a constant number of non-zero lattice vectors. Under the umbrella of identity-based cryptography [20], IBS eases the issue of authenticating public keys and may replace PKI systems in small domains.

An IBS scheme has a master public key Mpk and a master private key Msk. The scheme allows using an arbitrary string id, called identity, as a user's public key. Messages are signed by a private key $\mathsf{Sk}_{\mathsf{id}}$, extracted using Msk and the identity id. Verifying the signatures produced by $\mathsf{Sk}_{\mathsf{id}}$ is done publicly using Mpk, id. The standard security for IBS is existential unforgeability under chosen-identity and chosen-message attacks. This security notion is defined via a security game. In the security game, the adversary can make key extraction queries to request private keys for any identities of its choice. The adversary can also make signing queries to request signatures on identity/message pairs. Finally, the adversary breaks the IBS scheme if it comes up with a valid message/signature pair (m^*, σ^*) under a challenge identity id^* provided the adversary has not obtained $\mathsf{Sk}_{\mathsf{id}^*}$ and a valid signature on m^* under id^*. To get tight security, a reductionist proof is required with the following two probabilities are constants close to 1:

1. the probability of successfully replying *all* key extraction and signing queries.
2. the probability that the adversary's forgery can be exploited to solve the underlying computational problem.

While post-quantum IBS schemes can be obtained from different approaches (e.g., via hierarchical identity-based encryption schemes, identification schemes, and generically from normal digital signature schemes), the ones with short signatures and tight security are rare: the lattice-based IBS schemes by Pan and Wagner [15,16] are the only schemes proposed to date.

Pan and Wagner obtained their IBS scheme in two steps. First, a weakly secure lattice IBS scheme is built based on lattice trapdoor delegation techniques [1,5]. The IBS scheme is weakly secure in the sense that the key extraction queries and signing queries must be committed before seeing the master public key. Second, two generic transforms are devised that tightly convert the weakly secure IBS scheme into IBS schemes with standard security. The IBS schemes by Pan and Wagner [15,16] fall short in several aspects. Firstly, they lack detailed security proofs in the quantum random oracle model (QROM), which are desirable for post-quantum cryptographic schemes [3][1]. QROM security proofs model cryptographic hash functions as truly random functions called quantum-accessible random oracles. It is assumed that a quantum adversary can access these random oracles via superpositions and obtains the output quantum states.

Secondly, the generic transforms devised in [15] add extra overhead to signatures and may complicate security analysis. More specifically, the standard-model transform introduced in [15,16] adds two randomnesses of lattice-based

[1] The authors sketch a QROM proof. Combined with their standard model transform, the resulting IBS may be proved tightly secure in the QROM. However, a detailed QROM proof remains missing.

chameleon hashing [5] and two security-parameter-sized pre-images of cryptographic hashing, respectively, to each signature from the weakly secure IBS scheme. The added overhead, especially the lattice-based chameleon hashing randomnesses, is certainly not negligible (please refer to Sect. 1.4 for details.). Meanwhile, the random-oracle-model transform does require extra analysis in the QROM due to the extra random oracles. No QROM analysis is provided to the random-oracle-model transform in [15].

We note that Lee et al. [12] show that the generic certification approach gives tightly secure IBS schemes using a digital signature scheme that is tightly secure in the multi-user setting with adaptive corruption [7]. The only known such signature scheme, due to Pan and Wagner [17], needs the signature to contain a public key with size quadratic in the security parameter (see Table 8, [17]), resulting in IBS scheme no longer having short signatures.

In summary, the current state motivates constructing simple, more efficient, short post-quantum IBS schemes with tight security reductions in the QROM.

1.1 Our Contributions

We construct a quantumly secure IBS scheme with short signatures and a tight security reduction, improving the schemes from [15,16]. We give a detailed *tight* reductionist proof, which reduces the hardness of the standard short integer solution (SIS) problem to the strong unforgeability[2] of the IBS scheme in the quantum random oracle model (QROM). QROM security proofs model cryptographic hash functions as truly random functions called quantum-accessible random oracles. It is assumed that a quantum adversary can access these random oracles via superpositions and obtains the output quantum states.

Our IBS scheme is also provably and tightly secure in the classic random oracle model. The proof is similar to the QROM proof, and we give the proof sketch in Sect. 5. Like the IBS schemes from [15], our IBS scheme can be adapted to lattices over polynomial rings with the potential benefit of having shorter key sizes. This comes from the fact that the integer lattice tools that we use (e.g., lattice trapdoor delegation and preimage samplings) are all adaptable to lattices over polynomial rings [13].

1.2 Our Techniques

We give an overview of the tightly secure lattice-based IBS scheme by Pan and Wagner [15,16], referred to as the PW scheme. The PW scheme is constructed in two steps. A weakly and tightly secure IBS scheme is constructed first. In the security proof of the scheme, the adversary has to announce its key extraction queries and signing queries before seeing the public key. The master public key of the scheme contains a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and two hash functions $H_1 : \{0,1\}^* \to$

[2] Strong unforgeability stipulates that even with a signature of the message, the adversary still cannot derive a new signature on that message. Unforgeability prevents the adversary from producing a new message and its valid signature.

$\mathbb{Z}_q^{n\times m}$ and $H_2 : \{0,1\}^* \to \mathbb{Z}_q^{n\times m}$ modelled as random oracles. The master private key is a trapdoor matrix $\mathbf{R}$ that efficiently solves the SIS problem defined by $\mathbf{A}$. A private key for identity id is a short lattice basis $\mathbf{T}_{\mathsf{id}}$ for the matrix $[\mathbf{A}|H(\mathsf{id})]$, obtained by the master secret key $\mathbf{R}$ via lattice trapdoor delegation [1,5,13]. To sign a message m with identity id, a short Gaussian vector $\mathbf{d}$ is computed using $\mathbf{T}_{\mathsf{id}}$ such that $[\mathbf{A}|H_1(\mathsf{id})|H_2(\mathsf{id},\mathsf{m})]\mathbf{d} = \mathbf{0}$. In the security proof, the random oracles H_1 and H_2 are programmed using the adversary's committed queries, ensuring all queries can be answered without failure, making the proof tight. Second, two generic transforms are designed to tightly lift the weakly secure scheme into fully secure IBS schemes.

Our scheme achieves tight security very differently. We use the Katz-Wang random-bit technique [10] and the properties of preimage-samplable functions (PSF) [6]. The master public key contains a matrix $\mathbf{A}$ and two hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_q^{n\times m}$ and $H_2 : \{0,1\}^* \to \mathbb{Z}_q^{n}$, which are modelled as (quantum) random oracles. Given an identity id, a random bit $b \in \{0,1\}$ is selected. The identity key $\mathsf{Sk}_{\mathsf{id}} := (\mathbf{R}_{\mathsf{id}||b}, b)$ where $\mathbf{R}_{\mathsf{id}||b}$ is a lattice trapdoor for the matrix $[\mathbf{A}|H_1(\mathsf{id}||b)]$ and '$||$' denotes string concatenation. A signature on the identity id and message m is $\sigma := (\mathbf{d}, b)$ where $\mathbf{d}$ is a short Gaussian vector $\mathbf{d}$ such that $[\mathbf{A}|H_1(\mathsf{id}||b)]\mathbf{d} = H_2(\mathsf{id}||\mathsf{m})$, and b is the bit value attached to the private key $\mathsf{Sk}_{\mathsf{id}}$. Here $\mathbf{d}$ is produced via preimage sampling [1,6,13]. We stress that in the real scheme, both keys $\mathsf{Sk}_{\mathsf{id}} := (\mathbf{R}_{\mathsf{id}||b}, b)$ and $\mathsf{Sk}_{\mathsf{id}} := (\mathbf{R}_{\mathsf{id}||1-b}, 1-b)$ can be constructed using Msk, but only one of them is given.

Making the change on $H_2(\mathsf{id}||\mathsf{m})$ from the PW scheme for our scheme has positive effects. First, a signature of our scheme contains a vector of a lattice with a smaller dimension (smaller by $n \log q$). This makes our signature size 25% to 30% shorter than those from [15]. Second, a shorter signature size allows a weaker SIS assumption (since the security proof can extract better SIS problem solutions). Third, it enables stronger unforgeability. We compare our scheme with the IBS schemes in [15,16] in Sect. 1.4.

Our scheme is not subject to the technical difficulty of using the Katz-Wang random-bit technique for tightly secure IBS mentioned by Pan and Wagner [15]. This is because such a technical difficulty stems from using the random-bit technique to partition both the identity space *and* the message space. In contrast, our scheme only partitions the identity space.

1.3 Overview of Our Security Proof

We sketch the security reductionist proof in the ROM to show our ideas. For simplicity, we assume that identical adversarial queries will receive the same reply in the security reduction. For example, making a key extraction query on the identity id twice will receive the same identity key. In our full scheme, we use the standard technique, i.e., (hash-based) pseudo-random functions, to maintain such consistency without making the scheme stateful.

The reduction starts with a SIS problem instance defined by a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and uses $\mathbf{A}$ to simulate the public parameters to the hypothetical IBS adversary. For each identity id, the reduction selects a random bit $\mu \in \{0,1\}$.

The random oracle query $\mathsf{id}||b$ to H_1 is answered by programming $H_1(\mathsf{id}||b)$ in a way that $H_1(\mathsf{id}||b)$ is uniformly random, and $[\mathbf{A}|H(\mathsf{id}||b)]$ has a lattice trapdoor if $b = \mu$ but has no lattice trapdoors if $b = 1 - \mu$.

Given a key extraction query on the identity id, the reduction programs the random oracle output $H_1(\mathsf{id}||\mu)$ and uses the trapdoor to get the identity key $\mathsf{Sk}_{\mathsf{id}||\mu}$. It is crucial that μ looks random to the adversary given $\mathsf{Sk}_{\mathsf{id}||\mu}$. Note, the reduction cannot produce the other key $\mathsf{Sk}_{\mathsf{id}||1-\mu}$. Only one of the two keys is given in the real scheme. To answer a signing query on message m identity id, the reduction returns a stored signature if it already exists. Otherwise, the reduction programs $H_2(\mathsf{id}||\mathsf{m})$ as follows.

1. Select a random bit b and get $H_1(\mathsf{id}||b)$ by calling the random oracle H_1.
2. Sample a Gaussian vector $\mathbf{d}$, program $H_2(\mathsf{id}||\mathsf{m})$ as $[\mathbf{A}|H_1(\mathsf{id}||b)]\mathbf{d}$, and return $(\mathbf{d}, b)$ as the signature.

We do the same to answer the random oracle queries to H_2. Given a query $\mathsf{id}||\mathsf{m}$ to H_2, the above steps are performed to program $H_2(\mathsf{id}||\mathsf{m})$ and then store the signature for future queries. Applying the regularity argument from [6], $H_2(\mathsf{id}||\mathsf{m})$ is distributed statistically close to the uniform distribution over $\mathbb{Z}_q^n$. The signature is distributed as if it was generated correctly using one of the (unknown) identity keys, either $\mathsf{Sk}_{\mathsf{id}} = (\mathbf{R}_{\mathsf{id}||\mu}, \mu)$ or $\mathsf{Sk}_{\mathsf{id}} = (\mathbf{R}_{\mathsf{id}||1-\mu}, 1 - \mu)$, as required. Thus, security reduction can always answer signing queries.

Finally, the security reduction exploits the adversary's forged signature, e.g., $\sigma^* = (\mathbf{d}^*, \delta^*)$, on the identity id^* and message m^* where $\mathbf{d}^*$ is of low-norm and $[\mathbf{A}|H_1(\mathsf{id}||\delta^*)]\mathbf{d}^* = H_2(\mathsf{id}^*||\mathsf{m}^*)$. We assume a query $H_2(\mathsf{id}^*||\mathsf{m}^*)$ is made before the forgery is produced (without querying $H_2(\mathsf{id}^*||\mathsf{m}^*)$, σ^* will be valid with negligible probability). Recall that id^* is associated with a random value μ^*, the reduction can produce the identity key for id^* and δ^* when $\delta^* = \mu^*$, and can use the forgery when $\delta^* = 1 - \mu^*$. To exploit such a forgery, we need:

1. The internal states used to program $H_2(\mathsf{id}^*||\mathsf{m}^*)$, i.e., the value b^*, $\mathbf{d}$ such that $[\mathbf{A}|H_1(\mathsf{id}^*||b^*)]\mathbf{d} \to H_2(\mathsf{id}^*||\mathsf{m}^*)$.
2. The internal states used to simulate $H_1(\mathsf{id}||b^*)$.
3. Crucially, μ^* and δ^* that satisfy $b^* = 1 - \mu^*$ and $\delta^* = 1 - \mu^*$.

We argue that identity-associated bit μ^* remains from the adversary's view. First, the adversary cannot make a key extraction query on id^*. So, it cannot obtain μ^* from $\mathsf{Sk}_{\mathsf{id}^*}$. Moreover, μ^* is not leaked via the signing queries since $(\mathsf{id}^*, \mathsf{m}^*)$ is not allowed to ask for a signature. And, according to the reduction, the signing query on $(\mathsf{id}^*, \mathsf{m} \neq \mathsf{m}^*)$ only gets a random b^* from the signature, which is independent of μ^*. Meanwhile, μ^* is not used anywhere else in the reduction. Thus, the conditions $b^* = 1 - \mu^*$ and $\delta^* = 1 - \mu^*$ are met with a probability of $1/4$. So, the reduction solves the SIS problem with a probability close to $\epsilon/4$, making the reduction tight.

To prove the security in the QROM, we follow the works by Boneh et al. [3] and Katsumata et al. [9]. We carefully maintain the internal states to make the proof historic-free, meaning that the random oracle query responses are made without looking up previously generated values and states.

1.4 Comparison

We compare our scheme with the only known tightly secure and short lattice IBS schemes by Pan and Wagner [15,16] in Table 1. The IBS schemes from [15] are constructed by applying the two generic transforms to the weakly secure PW IBS scheme. The first transform does not require random oracles. It uses the lattice chameleon hash function [5]. The second one is in the ROM. We denote the IBS schemes obtained via the first and second transforms by PW'21-I and PW'21-II, respectively. The comparison shows that our IBS scheme outperforms essentially in all aspects, including relying on a weaker SIS assumption, enjoying shorter key and signature sizes, and achieving stronger unforgeability.

Table 1. Comparison among tightly secure and short IBS schemes from latices

Schemes	SIS param. β	$\lvert\mathsf{Mpk}\rvert$	$\lvert$Signature$\rvert$	Strongly Unforgeable	Proof in QROM
PW'21-I [15]	$\tilde{O}(n^{2.5})$	$(2nm + 2\lambda) \times V_{\mathbb{Z}_q}$	$3 \times V_{\mathbb{Z}^m} + 2 \times V_{\mathbb{Z}^w}$	NO	YES
PW'21-II [16]	$\tilde{O}(n^{2.5})$	$nm \times V_{\mathbb{Z}_q}$	$2 \times V_{\mathbb{Z}^{m+w}} + 4\lambda$	NO	NO
This work	$\tilde{O}(n^{1.5})$	$nm \times V_{\mathbb{Z}_q}$	$1 \times V_{\mathbb{Z}^{m+w}} + 1$ bit	YES	YES

Let λ be the security parameter. The parameters n, m, q, $w := n\lceil \log q \rceil$ are standard in the context of lattice-based cryptography [13]. The three schemes in Table 1 are set with the same n, m, q. The parameter β represents the strength of the underlying SIS problem. The larger β becomes, the easier the SIS problem becomes; hence, the less secure the scheme becomes.[3] The $\tilde{O}$ notation hides the logarithmic factors. Table 1 shows that our scheme uses a more dificult SIS problem, which is equivalent to saying that our scheme relies on a weaker SIS assumption.

We remark that the sizes of the master private key Msk and the identity key $\mathsf{Sk}_{\mathsf{id}}$ of the schemes are respectively dominated by the size of the lattice trapdoor matrices in $\mathbb{Z}^{m \times w}$ and $\in \mathbb{Z}^{(m+w) \times w}$. In addition, our scheme's master private and private identity keys, respectively contain two and one double-security-parameter-sized random binary strings. Since we are working on the (quantum) random oracle model, these random strings can be efficiently and deterministically generated by hash-based pseudorandom functions (see Lemma 2) using Msk and $\mathsf{Sk}_{\mathsf{id}}$ and barely affect the tightness of the reductions. Therefore, the schemes have essentially the same sizes in master private and identity keys.

The size of the master public key and signature are denoted by $\lvert\mathsf{Mpk}\rvert$ and $\lvert$Signature$\rvert$, respectively. $V_{\mathbb{Z}_q}$ and $V_{\mathbb{Z}^\ell}$ represent the size of a $\mathbb{Z}_q$-vector and a $\mathbb{Z}^\ell$-vector, respectively.[4] The quantity $2\lambda \times V_{\mathbb{Z}_q}$ appeared under $\lvert\mathsf{Mpk}\rvert$ is from

[3] The parameter β for PW'21-I and PW'21-II are from Sect. 4, [15].

[4] For the IBS schemes, the effect of the Gaussian variances of signatures on signature size is rather limited as the distributions are centred at zero.

the lattice-based chameleon hash function [5] used by the transform for PW'21-I. For a message $\mathbf{m} \in \{0,1\}^{\ell}$ and a randomness $\mathbf{r} \in \mathbb{Z}^m$, The chameleon hash function is defined as $CHF(\mathbf{m};\mathbf{r}) := \mathbf{Br} + \mathbf{Cm}$ where $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{C} \in \mathbb{Z}_q^{n \times \ell}$. The chameleon hash function takes at least 2λ-bit inputs for λ-bit security due to the quantum algorithm Grover's search [8]. So, we set $\ell = 2\lambda$. The signature of PW'21-II has two random strings that are input to the random oracles. To ensure λ-bit security, the strings are at least 2λ bits long due to Grover's search.

PW'21-I and PW'21-II are not strongly unforgeable as multiplying a signature vector by a small constant, say 2, will likely result in a valid signature vector. Finally, after the publication of [16] was updated with a sketched security proof of PW'21-I in the QROM. No QROM proof is given to PW'21-II.

2 Preliminaries

We denote the security parameter by λ. $\mathsf{poly}(\lambda)$ represents any polynomial in λ. We denote by $\mathsf{negl}(n)$ any negligible function in n. We use the standard asymptotic notations $O(\cdot)$, $\omega(\cdot)$, and $\Omega(\cdot)$. We use bold lowercase letters (e.g. $\mathbf{a}$) to denote vectors and bold capital letters (e.g. $\mathbf{A}$) to denote matrices. For an integer $q \geq 2$, let $\mathbb{Z}_q$ be the ring of integers modulo q. We denote by $[\mathbf{A}|\mathbf{B}]$ the concatenation of $\mathbf{A}$ and $\mathbf{B}$. We denote by $\|\mathbf{x}\|$ the ℓ_2 norm of a vector $\mathbf{x}$. $s_1(\mathbf{X})$ denotes the spectrum norm of the matrix $\mathbf{R}$, i.e., $s_1(\mathbf{R}) = \max_{\|\mathbf{u}\|=1} \|\mathbf{Xu}\|$.

If X is some distribution, we denote by $x \leftarrow X$ a sample x drawn according to the distribution X. We denote by $x \leftarrow U(\mathcal{X})$ drawing a sample x uniformly at random from a finite set $\mathcal{X}$. Let X and Y be two random variables taking values in some finite set S. We say random variables (or distributions) are statistically close or $\mathsf{negl}(\lambda)$-close if their statistical distance, $\Delta(X,Y) := \frac{1}{2}\sum_{s \in S} |\Pr[X = s] - \Pr[Y = s]|$ is negligible in the security parameter λ.

2.1 Quantum Computation

We follow a series of existing works [3,9,11,19] about cryptography in QROM to give a brief overview of quantum computation. A n qubits state $|\psi\rangle$ is expressed as $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle \in \mathbb{C}^{2^n}$ where $\{\alpha_x\}_{x \in \{0,1\}^n}$ satisfies $\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$ is a set of complex numbers. $\{|x\rangle\}_{x \in \{0,1\}^n}$ is called an orthonormal computational basis. An n-qubit state $|\psi\rangle$ can be measured using the computational basis, The measurement results in a classic bit string $x \in \{0,1\}^n$ with probability $|\alpha_x|^2$ and after the measurement, the state becomes $|x\rangle$. An evolution of quantum state can be described by a unitary matrix $\mathbf{U}$, which transforms $|x\rangle$ to $U|x\rangle$. A quantum algorithm is composed of quantum evolutions described by unitary matrices and measurements. Given any classicly computatble function f, there exists an unitary matrix $\mathbf{U}_f$ such that $\mathbf{U}_f|x,y\rangle = |x, f(x) \oplus y\rangle$. The running time of a quantum algorithm $\mathcal{A}$ is defined as the number of universal gates and measurements required for running $\mathcal{A}$.

The quantum random oracle model (QROM) is an idealised model in which a cryptographic hash function is modelled as a truly random function that can be accessed publicly and quantumly. In the security proof in the QROM, a random function $H : X \to Y$ is selected uniformly at random at the beginning of the security experiment (or security games). Every entity involved in the experiment is given access to an oracle, which given a state $\sum_{x,y} \alpha_{x,y} |x, y\rangle$ returns $\sum_{x,y} \alpha_{x,y} |x, H(x) \oplus y\rangle$. In the QROM, one query to the oracle is counted as one unit of time. As mentioned in [9], one random function H can be used to implement n random functions for n quantum (and classical) random oracles: for integer $0 \leq i \leq n-1$, the i-th random function is defined as $H_i(x) := H(i||x)$. We need the following two lemmas due to Boneh et al. [3] and Saito et al. [19].

Lemma 1. *Let $\epsilon > 0$. Say $\mathcal{A}$ is a quantum algorithm that makes Q quantum oracle queries. Suppose that we draw the oracle O from two distributions. The first is the random oracle distribution. The second is the distribution of oracles where the value of the oracle at each input x is identically and independently distributed by some distribution D whose statistical distance is within ϵ from uniform. Then the statistical distance between the distributions of outputs of $\mathcal{A}$ with each oracle is at most $4Q^2\sqrt{\epsilon}$.*

Lemma 2. *Let λ be the security parameter. Let ℓ be a positive integer. Let $H : \{0,1\}^\ell \times X \to Y$ and $H' : X \to Y$ be two independent random oracles. If an unbounded time adversary $\mathcal{A}$ who is given quantum access to H and H' makes at most Q_H queries to to H, then for $k \leftarrow U(\{0,1\})^\ell$ we have*

$$\left| \Pr[\mathcal{A}^{|H\rangle, |H(k,\cdot)\rangle}(1^\lambda) = 1] - \Pr[\mathcal{A}^{|H\rangle, |H'\rangle}(1^\lambda) = 1] \right| \leq 2Q_H \cdot 2^{-\ell/2}$$

According to Saito et al., [19], the above lemma also applies to the ROM, i.e., H and H' are classic random oracles.

2.2 Lattices and Discrete Gaussians

Definition 1. *For a positive integer q (later to be prime), a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, define*

$$\Lambda_q^\perp(\mathbf{A}) := \{\mathbf{y} \in \mathbb{Z}^m \ s.t.\ \mathbf{A}\mathbf{y} = \mathbf{0} \pmod{q}\}$$

as the m-dimensional full-rank integer lattices and

$$\Lambda_q^{\mathbf{u}}(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m \ : \mathbf{A}\mathbf{e} = \mathbf{u} \pmod{q}\}$$

for $\mathbf{u} \in \mathbb{Z}_q^n$ as the shift of $\Lambda_q^\perp(\mathbf{A})$.

Definition 2. *Let $m \in \mathbb{Z}$ be a positive integer and $\Lambda \subseteq \mathbb{Z}^m$. For any real vector $\mathbf{c} \in \mathbb{R}^m$ and positive parameter $\sigma \in \mathbb{R}$, $\forall \mathbf{y} \in \Lambda$, the discrete Gaussian distribution over Λ with centre $\mathbf{0}$ and parameter σ is defined by $D_{\Lambda,\sigma} = \frac{\rho_\sigma(\mathbf{y})}{\rho_\sigma(\Lambda)}$ where $\rho_\sigma(\mathbf{x}) = \exp\left(-\pi\|\mathbf{x}\|^2/\sigma^2\right)$ is the Gaussian function and $\rho_\sigma(\Lambda) = \sum_{\mathbf{x} \in \Lambda} \rho_\sigma(\mathbf{x})$.*

Lemma 3. (Special case of Lemma 2.9, [13]). *Let n, m be positive integers, and $\mathbf{R} \leftarrow D_{\mathbb{Z},s}^{n\times m}$. There is a constant $C \approx 1/\sqrt{2\pi}$ such that for any $t \geq 0$, $s_1(\mathbf{R}) \leq C \cdot s \cdot (\sqrt{n} + \sqrt{m} + t)$ except with probability at most $2\exp(-\pi t^2)$.*

Lemma 4. (Lemma 4.4 of [14]). *For any lattice $\Lambda \subseteq \mathbb{Z}^n$, and reals $0 < \epsilon < 1$, $r \geq \omega(\sqrt{\log n})$, we have $\Pr_{\mathbf{x}\sim D_{\Lambda,r,\mathbf{c}}}[\|\mathbf{x}\| > r\sqrt{n}] \leq \frac{1+\epsilon}{1-\epsilon} \cdot 2^{-n}$.*

Lemma 5. *For any n-dimensional lattice $\Lambda \subseteq \mathbb{Z}^n$, positive $\epsilon < 1/3$, and $s \geq \omega(\sqrt{\log n})$, and for every $\mathbf{x} \in \Lambda$, the min-entropy $\mathrm{H}_\infty(D_{\Lambda,s}) \geq n - 1$.*

2.3 Lattice Trapdoors

Let $k = \lceil \log q \rceil$, $\mathbf{g}^\intercal = (1, 2, 4, \ldots, 2^{k-1})$. Define the n-by-nk gadget matrix as

$$\mathbf{G} = [\mathbf{g}^\intercal \otimes \mathbf{I}_n] = \begin{bmatrix} \mathbf{g}^\intercal & & & \\ & \mathbf{g}^\intercal & & \\ & & \ddots & \\ & & & \mathbf{g}^\intercal \end{bmatrix} \in \mathbb{Z}_q^{n\times nk}.$$

We recall the several lemmas that relates to the lattice trapdoor techniques. The lemma is a direct implication of Theorem 4.1.

Lemma 6. *There is a p.p.t algorithm $\mathsf{SampleR}(m, k, s)$ that takes as input $s \geq (\sqrt{\log n})$, and samples from a distribution statistically close to $D_{\mathbb{Z}^m,s}^k$.*

Let n, m, q, k be positive integers, where $m \geq 2n\log q + \omega(\sqrt{\log n})$. Let $w = \lceil n \log q \rceil$, $r = \omega(\sqrt{\log n})$. The first lemmas below are from Algorithm 1, Algorithm 3, and Algorithm 4 of [13]. The last lemma is from Proposition 5.1, [6].

Lemma 7. *There is a p.p.t algorithm $\mathsf{TrapGen}(1^n, 1^m, q)$ returns a matrix $\mathbf{A} = [\bar{\mathbf{A}}|\bar{\mathbf{A}}\mathbf{R} + \mathbf{H}\mathbf{G}] \in \mathbb{Z}_q^{n\times m}$ for $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n\times(m-w)}$ and invertible $\mathbf{H} \in \mathbb{Z}_q^{n\times n}$ such that $\mathbf{A}$'s distribution is statistically close to $U(\mathbb{Z}_q^{n\times m})$ and the matrix $\mathbf{R}$'s distribution is statistically close to $D_{\mathbb{Z},r}^{(m-w)\times w}$. $\mathbf{R}$ is called a $\mathbf{G}$-trapdoor of $\mathbf{A}$.*

Lemma 8. *There is a p.p.t algorithm $\mathsf{SampleD}(\mathbf{A}, \mathbf{R}, \mathbf{U}, s)$ takes as input a $\mathbb{Z}_q^{n\times m}$-matrix $\mathbf{A}$ along with its $\mathbf{G}$-trapdoor $\mathbf{R}$, a vector $\mathbf{U} \in \mathbb{Z}_q^{n\times k}$ and $s \geq \omega(\sqrt{\log n}) \cdot s_1(\mathbf{R})$, and returns a vector $\mathbf{d} \in \mathbb{Z}^m$ which has a distribution statistically close to $D_{\mathbb{Z}^m,r}^k$ conditioned on $\mathbf{A}\mathbf{d} = \mathbf{U} \pmod q$, i.e., $D_{\Lambda_q^{\mathbf{U}}(\mathbf{A}),s}$.*

Lemma 9. *There is a p.p.t algorithm $\mathsf{DelTrap}(\mathbf{A}, \mathbf{R}, \mathbf{A}', \mathbf{H}, s')$ takes as input a $\mathbb{Z}_q^{n\times m}$-matrix $\mathbf{A}$, a $\mathbf{G}$-trapdoor $\mathbf{R}$ of $\mathbf{A}$, a $\mathbb{Z}_q^{n\times w}$-matrix $\mathbf{A}'$, and a Gaussian parameter s' where $s' \geq \omega(\sqrt{\log n}) \cdot s_1(\mathbf{R})$, and outputs a matrix $\mathbf{R}'$ whose distribution is statistically close to $D_{\mathbb{Z}^m,s'}^w$ conditioned on $\mathbf{A}' = \mathbf{A}\mathbf{R}' + \mathbf{H}\mathbf{G}$.*

Lemma 10. *For $\mathbf{A} \leftarrow U(\mathbb{Z}_q^{n\times m})$ and $\mathbf{e} \leftarrow D_{\mathbb{Z}^m,s}$ with any $s \geq \omega(\sqrt{\log n})$, the distribution of $\mathbf{A}\mathbf{e} \bmod q$ is statistically close to $U(\mathbb{Z}_q^n)$. Furthermore, for a fixed $\mathbf{u} \in \mathbb{Z}_q^n$, the distribution of $\mathbf{e} \leftarrow D_{\mathbb{Z}^m,s}$ given $\mathbf{A}\mathbf{e} = \mathbf{u} \bmod q$ is $D_{\Lambda_q^{\mathbf{u}}(\mathbf{A}),s}$.*

Our construction uses a special cases of Lemma 7 and Lemma 9 by setting the matrix $\mathbf{H} \in \mathbb{Z}_q^{n\times n}$ as the identity matrix, i.e., $\mathbf{I}_n$. Throughout this paper, we make the randomness used by the above algorithms explicit when needed and separate them from the primary inputs by semicolons. For example, $\mathsf{SampleR}(m, k, s; \phi)$, $\mathsf{SampleD}(\mathbf{A}, \mathbf{R}, \mathbf{U}, s; \varphi)$ and $\mathsf{DelTrap}(\mathbf{A}, \mathbf{R}\ , \mathbf{A}', \mathbf{H}, s; \psi)$, where ϕ, φ, and ψ denote the randomness used by these algorithms.

2.4 Short Integer Solution (SIS) Problem

Definition 3. *Let λ be the security parameter. The advantage of an algorithm $\mathcal{A}$ that solves the $\mathsf{SIS}_{n,q,m,\beta}$ problem, defined as,*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SIS}_{n,q,m,\beta}}(\lambda) := \Pr\left[\mathcal{A}(\mathbf{A}, \beta) \rightarrow \mathbf{e} \neq \mathbf{0} \ : \ \mathbf{A}\mathbf{e} = 0 \bmod q \ \wedge \ \|\mathbf{e}\| \leq \beta\right]$$

where $\mathbf{A} \leftarrow U(\mathbb{Z}_q^{n\times m})$, We say (t, ϵ)-$\mathsf{SIS}_{n,q,m,\beta}$ assumption holds if for all t-time algorithms $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SIS}_{n,q,m,\beta}}(\lambda) \leq \epsilon$.

Our IBS scheme uses the $\mathsf{SIS}_{n,q,m,\beta}$ problem with prime q and $\beta = \tilde{O}(n^{3/2})$, which is far from being reached by the known classic and quantum algorithms. We refer to Sect. 2.1, [18] for detailed discussions.

2.5 Identity-Based Signatures

Let λ be the security parameter. An identity-based signature (IBS) scheme $\mathcal{IBS}$ consists of four probabilistic polynomial time (p.p.t) algorithms.

- $\mathsf{Setup}(1^\lambda)$ returns a master public key Mpk and a master private key Msk.
- $\mathsf{Ext}(\mathsf{Mpk}, \mathsf{Msk}, \mathsf{id})$ generates an identity private key $\mathsf{Sk}_{\mathsf{id}}$.
- $\mathsf{Sig}(\mathsf{Mpk}, \mathsf{Sk}_{\mathsf{id}}, \mathsf{id}, \mathsf{m})$ returns a signature σ on a message m and identity id.
- $\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}, \mathsf{m}, \sigma)$ returns 1, indicating the signature σ is valid for the message m from a signer whose identity is id, or 0, otherwise.

We require for standard correctness, i.e., for all $(\mathsf{Mpk}, \mathsf{Msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{Sk}_{\mathsf{id}} \leftarrow \mathsf{Ext}(\mathsf{Mpk}, \mathsf{Msk}, \mathsf{id})$,

$$\Pr\left[\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}, \mathsf{m}, \mathsf{Sig}(\mathsf{Mpk}, \mathsf{Sk}_{\mathsf{id}}, \mathsf{id}, \mathsf{m}) = 1\right] \geq 1 - \mathsf{negl}(\lambda)$$

where $\mathsf{negl}(\lambda)$ is negligible in λ.

We recall the strong existential unforgeability under the chosen-identity attack and the chosen-message attack (SEUF-ID-CMA) for IBS. The notion is strong in the sense that no adversary can forge a signature on a message-identity pair from which the adversary has seen valid signatures. A weaker unforgeability notion (EUF-ID-CMA) as considered in [15] only requires that a forgery cannot be produced for a message-identity pair that have not been signed.

```
Experiment $\mathsf{Exp}^{\mathrm{seuf}}_{\mathcal{IBS},\mathcal{A}}(\lambda)$:
1. $L_{\mathsf{id}} \leftarrow \emptyset$, $L_\sigma \leftarrow \emptyset$
2. $(\mathsf{Mpk}, \mathsf{Msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3. $(\mathsf{id}^*, \mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{OExt}(),\mathsf{OSig}(),\mathsf{RO}()}(\mathsf{Mpk})$
4. If $\mathsf{id}^* \in L_{\mathsf{id}}$, return 0
5. If $(\mathsf{id}^*, \mathsf{m}^*, \sigma^*) \in L_\sigma$, return 0
6. Return $\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}^*, \mathsf{m}^*, \sigma^*)$

Oracle $\mathsf{RO}()$:
Quantum or classic random oracle(s)
```

```
Oracle $\mathsf{OExt}(\mathsf{id})$:
1. Return $\mathsf{Sk}_{\mathsf{id}} \leftarrow \mathsf{Ext}(\mathsf{Mpk}, \mathsf{Msk}, \mathsf{id})$
2. $L_{\mathsf{id}} = \{\mathsf{id}\} \cup L_{\mathsf{id}}$

Oracle $\mathsf{OSig}(\mathsf{id}, \mathsf{m})$:
1. $\mathsf{Sk}_{\mathsf{id}} \leftarrow \mathsf{Ext}(\mathsf{Mpk}, \mathsf{Msk}, \mathsf{id})$
2. Return $\sigma \leftarrow \mathsf{Sig}(\mathsf{Mpk}, \mathsf{Sk}_{\mathsf{id}}, \mathsf{id}, \mathsf{m})$
3. $L_\sigma = \{\mathsf{id}, \mathsf{m}, \sigma\} \cup L_\sigma$
```

Fig. 1. SEUF-ID-CMA Security Experiment

Definition 4. *Consider the security experiment (game) defined in 1. We say that an IBS scheme $\mathcal{IBS}$ is $(t, Q_{\mathsf{Ext}}, Q_{\mathsf{Sig}}, \epsilon)$-SEUF-ID-CMA secure if for any t-time adversary $\mathcal{A}$ that makes at most Q_{Ext} (key extraction) queries to the oracle* $\mathsf{OExt}()$ *and at most Q_{Sig} (signing) queries to the oracle* $\mathsf{OSig}()$*, we have*

$$\mathsf{Adv}^{\mathsf{seuf}}_{\mathcal{IBS},\mathcal{A}}(\lambda) := \Pr[\mathsf{Exp}^{\mathsf{seuf}}_{\mathcal{IBS},\mathcal{A}}(\lambda) = 1] \leq \epsilon$$

where the probability is over the randomness of $\mathcal{A}$ and the IBS scheme.

3 Our IBS Scheme

We provide our IBS scheme Σ in Fig. 2. The scheme uses the following global parameters and cryptographic hash functions:

- Gadget matrix $\mathbf{G}$'s column dimension $w := n\lceil \log q \rceil$.
- Dimension $m \geq 2n \log q + \omega(\sqrt{\log n})$ for using Lemma 9.
- Gaussian parameter $r = \omega(\sqrt{\log n})$ for the algorithm $\mathsf{TrapGen}$.
- Gaussian parameter $s = O(\sqrt{w}) \cdot \omega(\sqrt{\log n})$ for algorithm $\mathsf{DelTrap}$.
- Gaussian parameter $s' = O(w) \cdot \omega(\sqrt{\log n})^2$ for algorithm $\mathsf{SampleD}$.
- Identity space $\{0,1\}^{\ell_{\mathsf{id}}}$ and message space $\{0,1\}^{\ell_{\mathsf{m}}}$ for $\ell_{\mathsf{id}}, \ell_{\mathsf{m}}$.
- ℓ_s and $\ell_{s'}$, the randomness bit lengths of $\mathsf{DelTrap}$ and $\mathsf{SampleD}$, respectively.
- Cryptographic hash functions modelled as (quantum) random oracles:
    - $\mathsf{H}_1 : \{0,1\}^\ell \times \{0,1\}^{\ell_{\mathsf{id}}} \rightarrow \{0,1\}$, $\mathsf{H}_2 : \{0,1\}^\ell \times \{0,1\}^{\ell_{\mathsf{id}}}\{0,1\}^{\ell_s}$, $\mathsf{H}_3 : \{0,1\}^\ell \times \{0,1\}^{\ell_{\mathsf{id}}+\ell_{\mathsf{m}}} \rightarrow \{0,1\}^{\ell_{s'}}$
    - $H_1 : \{0,1\}^{\ell_{\mathsf{id}}+1} \rightarrow \mathbb{Z}_q^{n\times w}$, $H_2 : \{0,1\}^{\ell_{\mathsf{id}}+\ell_{\mathsf{m}}} \rightarrow \mathbb{Z}_q^n$

In the scheme, each user (with identity id) also chooses and keeps a secret random string $k_{\mathsf{id}} \leftarrow U(\{0,1\}^\ell)$ for signing. $\mathsf{H}_1(k_1, \cdot)$, $\mathsf{H}_2(k_2, \cdot)$ and $\mathsf{H}_3(k_{\mathsf{id}}, \cdot)$ are used to make the signing and key extraction processes deterministic, i.e., if the same message and identity are submitted for signing (resp. key extraction), the same signature (resp. identity key) will be returned.

```
Setup($1^\lambda$):
1. $(\mathbf{A}, \mathbf{R}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$
2. $k_1, k_2 \leftarrow U(\{0,1\}^\ell)$
3. Return $\mathsf{Mpk} := \mathbf{A}$, $\mathsf{Msk} := \{\mathbf{R}, k_1, k_2\}$

Sig(Mpk, Sk_id, id, m):
*User id keeps a secret $k_{\mathsf{id}} \leftarrow U(\{0,1\}^\ell)$
1. Parse $\mathsf{Sk}_{\mathsf{id}} = (\mathbf{R}_{\mathsf{id}||b}, b)$
2. $\mathbf{u} \leftarrow H_2(\mathsf{id}||\mathsf{m})$
3. $\mathbf{A}_{\mathsf{id}||b} \leftarrow H_1(\mathsf{id}||b)$, $\mathbf{F}_{\mathsf{id}||b} := [\mathbf{A}|\mathbf{A}_{\mathsf{id}||b}]$
4. $\varphi \leftarrow \mathsf{H}_3(k_{\mathsf{id}}, \mathsf{id}||\mathsf{m})$
5. $\mathbf{d} \leftarrow \mathsf{SampleD}(\mathbf{F}_{\mathsf{id}||b}, \mathbf{R}_{\mathsf{id}||b}, \mathbf{u}, s'; \varphi)$
6. Return $\sigma := (\sigma_1, \sigma_2) = (\mathbf{d}, b)$

Ext(Mpk, Msk, id):
1. $b \leftarrow \mathsf{H}_1(k_1, \mathsf{id})$, $\psi \leftarrow \mathsf{H}_2(k_2, \mathsf{id})$
2. $\mathbf{A}_{\mathsf{id}||b} \leftarrow H_1(\mathsf{id}||b)$
3. $\mathbf{R}_{\mathsf{id}||b} \leftarrow \mathsf{DelTrap}(\mathbf{A}, \mathbf{R}, \mathbf{A}_{\mathsf{id}||b}, \mathbf{I}_n, s; \psi)$
4. Return $\mathsf{Sk}_{\mathsf{id}} := (\mathbf{R}_{\mathsf{id}||b}, b)$

Ver(Mpk, id, $\sigma$, m):
1. Parse $\sigma := (\mathbf{d}, b)$
2. $\mathbf{u} \leftarrow H_2(\mathsf{id}||\mathsf{m})$
3. Return 0 if $[\mathbf{A}|H_1(\mathsf{id}||b)]\mathbf{d} \neq \mathbf{u}$
4. Return 0 if $\|\mathbf{d}\| > s'\sqrt{m+w}$
5. Return 1
```

Fig. 2. The Proposed IBS Scheme

Correctness. We show that $\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}, \mathsf{m}, \mathsf{Sig}(\mathsf{Mpk}, \mathsf{Sk}_{\mathsf{id}}, \mathsf{id}, \mathsf{m}) = 1$ happens with all but negligible probability. By Lemma 8, $\mathsf{Sig}(\mathsf{Mpk}, \mathsf{Sk}_{\mathsf{id}}, \mathsf{id}, \mathsf{m})$ returns a signature $\sigma := (\mathbf{d}, b)$ where $\mathbf{d}$ distributes statistically close to $D^{m+w}_{\mathbb{Z},s'}$, conditioned on $[\mathbf{A}|H_1(\mathsf{id}||b)]\mathbf{d} = H_2(\mathsf{id}||\mathsf{m})$. So, step 3 of the algorithm $\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}, \sigma, \mathsf{m})$ returns 0 with negligible probability. Meanwhile, by Lemma 4, $\|\mathbf{d}\| \leq s'\sqrt{m+w}$ with all but negligible probability, which proves that $\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}, \sigma, \mathsf{m})$ returns 1, i.e., accepts the correctly generated signature, with all but negligible probability.

4 Security Proof in the Quantum Random Oracle Model

Theorem 1. *Let H_1, H_2, H_1, H_2, H_3 be random oracles that may be quantumly accessible. If (t, ϵ)-$\mathsf{SIS}_{n,q,m,\beta}$ assumption holds with $\beta = O(w^{3/2}) \cdot \omega(\sqrt{\log n})^2$, the IBS scheme $\varSigma$ is $(t', Q_{\mathsf{Ext}}, Q_{\mathsf{Sig}}, 4\epsilon + \mathsf{negl}(\lambda))$-SEUF-ID-CMA secure for arbitrary Q_{Ext}, Q_{Sig}, and any $t' \leq t + O(Q_{H_1} + Q_{H_2}) \cdot \mathsf{poly}(\lambda)$ where Q_{H_1} and Q_{H_2} are the number of quantum random oracle queries to $|H_1\rangle$ and $|H_2\rangle$, respectively.*

We split our security proof into two steps. First, we devise a stateful version of our original IBS scheme $\varSigma$ shown in Fig. 2, denoted by $\varSigma'$, which preserves the security from $\varSigma$. Then, we reduce the hardness of $\mathsf{SIS}_{a,q,n,\beta}$ problem to the SEUF-ID-CMA security of the stateful IBS scheme $\varSigma'$.

4.1 Security Reduction from IBS Scheme $\varSigma'$ to IBS Scheme $\varSigma$

The stateful scheme, as described in Fig. 3, removes H_1, H_2, H_3, and uses internal states to make consistent replies to repeated key extracting and signing queries. The (pseudo)random values produced by H_1, H_2, H_3 in the original scheme are instead sampled uniformly at random from the corresponding sets.

Considering the SEUF-ID-CMA security experiment (Fig. 1), it is clear that in both the original IBS scheme $\varSigma$ and the stateful IBS scheme $\varSigma'$, the same query id to OExt is responded by the same identity key, and the same query

```
$\mathsf{Setup}(1^\lambda)$:
    $(\mathbf{A}, \mathbf{R}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$
    Return $\mathsf{Mpk} := \mathbf{A}$, $\mathsf{Msk} := \mathbf{R}$
$^*$Each signer initialises sets $L_{\mathsf{Sig}}, L_{\mathsf{Sk}} \leftarrow \emptyset$
$\mathsf{Sig}(\mathsf{Mpk}, \mathsf{Sk}_{\mathsf{id}}, \mathsf{m})$:
    Return $\sigma$ if $(\mathsf{id}, \mathsf{m}, \sigma) \in L_{\mathsf{Sig}}$
    Parse $\mathsf{Sk}_{\mathsf{id}} = (\mathbf{R}_{\mathsf{id}||b}, b)$
    $\mathbf{u} \leftarrow H_2(\mathsf{id}||\mathsf{m})$
    $\mathbf{A}_{\mathsf{id}||b} \leftarrow H_1(\mathsf{id}||b)$, $\mathbf{F}_{\mathsf{id}||b} := [\mathbf{A}|\mathbf{A}_{\mathsf{id}||b}]$
    $\varphi \leftarrow U(\{0,1\}^{\ell_{s'}})$
    $\mathbf{d} \leftarrow \mathsf{SampleD}(\mathbf{F}_{\mathsf{id}||b}, \mathbf{R}_{\mathsf{id}||b}, \mathbf{u}, s'; \varphi)$
    $L_{\mathsf{Sig}} \leftarrow \{(\mathsf{id}, \mathsf{m}, \sigma)\} \cup L_{\mathsf{Sig}}$
    Return $\sigma := (\sigma_1, \sigma_2) := (\mathbf{d}, b)$

$\mathsf{Ext}(\mathsf{Mpk}, \mathsf{Msk}, \mathsf{id})$:
    Return $\mathsf{Sk}_{\mathsf{id}}$ if $(\mathsf{id}, \mathsf{Sk}_{\mathsf{id}}) \in L_{\mathsf{Sk}}$
    $b \leftarrow U(\{0,1\})$, $\psi \leftarrow U(\{0,1\}^{\ell_s})$
    $\mathbf{A}_{\mathsf{id}||b} \leftarrow H_1(\mathsf{id}||b)$
    $\mathbf{R}_{\mathsf{id}||b} \leftarrow \mathsf{DelTrap}(\mathbf{A}, \mathbf{R}, \mathbf{A}_{\mathsf{id}||b}, s; \psi)$
    $L_{\mathsf{Sk}} \leftarrow \{(\mathsf{id}, \mathsf{Sk}_{\mathsf{id}})\} \cup L_{\mathsf{Sk}}$
    Return $\mathsf{Sk}_{\mathsf{id}} := (\mathbf{R}_{\mathsf{id}||b}, b)$
$\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}, \sigma, \mathsf{m})$:
    Parse $\sigma := (\mathbf{d}, b)$
    $\mathbf{u} \leftarrow H_2(\mathsf{id}||\mathsf{m})$
    Return 0 if $[\mathbf{A}|H_1(\mathsf{id}||b)]\mathbf{d} \neq \mathbf{u}$
    Return 0 if $\|\mathbf{d}\| > s'\sqrt{m+w}$
    Return 1
```

Fig. 3. The Stateful IBS Scheme Σ'

$(\mathsf{id}, \mathsf{m})$ to OSig is responded with the same signature value σ. Therefore, the views of the adversary $\mathcal{A}$ to Σ and Σ' are identical except how $b \in \{0,1\}$ and the randomness for $\mathsf{SampleD}$ and $\mathsf{DelTrap}$ are generated, i.e., generated using H_1, H_2, H_3 versus generated uniformly at random. We note that queries to OSig trigger queries to $\mathsf{H}_1(k_1, \cdot)$, and queries to OExt trigger queries to $\mathsf{H}_2(k_2, \cdot)$ and $\mathsf{H}_3(k_3, \cdot)$[5]. By Lemma 2 and union bound, a simple reduction shows

$$|\mathsf{Adv}^{\mathsf{seuf}}_{\Sigma,\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{seuf}}_{\Sigma',\mathcal{A}}(\lambda)| \leq 2(2Q_{\mathsf{Ext}} + Q_{\mathsf{Sig}}) \cdot 2^{-\ell/2} \tag{1}$$

which is negligible in λ for polynomial ℓ (in λ), Q_{Ext} and Q_{Sig}. The reduction runs in time nearly the same as the run time of the adversary against Σ plus the time taken to simulate the parameters of Σ, which is polynomial in λ.

4.2 The Security of IBS Scheme Σ'

Proof. Assume there is a t-time SEUF-ID-CMA adversary who makes at most Q_{H_1} and Q_{H_2} queries, respectively to $|H_1\rangle$ and $|H_2\rangle$ and has advantage $\mathsf{Adv}^{\mathsf{seuf}}_{\Sigma',\mathcal{A}}(\lambda)$ in breaking the IBS Scheme Σ'. We use algorithm $\mathsf{SampleR}(k_1, k_2, s; \mathsf{coin})$ whose output distribution is statistically close to $D^{k_2}_{\mathbb{Z}^{k_1},s}$. coin denotes the randomness: when $k_1 = m$, $k_2 = w$, $|\mathsf{coin}| = \ell_c$; when $k_1 = m + w$, $k_2 = 1$, $|\mathsf{coin}| = \ell_{c'}$.

We proceed with a sequence of security games defined in Fig. 4. Three procedures can describe each security game: Initialisation (defined in Fig. 5), Setup (defined in Fig. 6), and Ver. Each security game has four oracles: the key extraction oracle OExt (defined in Fig. 10), the signing oracle OSig (defined in Fig. 9); and two quantum random oracles $|H_1\rangle$, $|H_2\rangle$ (defined in Fig. 7 and Fig. 8, respectively). Ver is identical to the verification algorithm given in Fig. 2 and is kept the same across all games. The process Initialisation is included in the proof where random functions are set as random oracles accessed by the participants.

Let S_i be the event that the ith security game returns a well-defined value 1. The first security game G_0 runs the stateful IBS scheme Σ' according to the

[5] The adversary does not have direct oracle access to $\mathsf{H}_1(k_1, \cdot)$, $\mathsf{H}_2(k_2, \cdot)$ and $\mathsf{H}_3(k_3, \cdot)$.

```
$G_i$ for $i = 0, 1, ..., 5$:
1. Initialising $L_{\mathsf{id}} \leftarrow \emptyset$, $L_\sigma \leftarrow \emptyset$
2. $(\mathsf{Mpk}, \mathsf{Msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3. $(\mathsf{id}^*, \mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{OExt}, \mathsf{OSig}, |H_1\rangle, |H_2\rangle}(\mathsf{Mpk})$
4. If $\mathsf{id}^* \in L_{\mathsf{id}}$, return 0
5. If $(\mathsf{id}^*, \mathsf{m}^*, \sigma^*) \in L_\sigma$, return 0
6. Return $\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}^*, \mathsf{m}^*, \sigma^*)$

Initialisation:
Initialising quantum random oracle(s);
As defined in Fig. 5

Procedure $\mathsf{Setup}(1^\lambda)$:
As defined in Fig. 6
Oracles $|H_1\rangle()$:
Quantum RO as defined in Fig. 7
Oracle $|H_2\rangle()$:
Quantum RO as defined in Fig. 8
Oracle $\mathsf{OExt}(\mathsf{id})$:
As defined in Fig. 10
Oracle $\mathsf{OSig}(\mathsf{id}, \mathsf{m})$:
As defined in Fig. 9
```

Fig. 4. Security games G_i for proofs in QROM

```
Initialisation in $G_0$:
1. $H_1 \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}+1} \rightarrow \mathbb{Z}_q^{n \times w})$
2. $H_2 \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}+\ell_{\mathsf{m}}} \rightarrow \mathbb{Z}_q^n)$
Initialisation in $G_1$:
1. $H_2 \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}+\ell_{\mathsf{m}}} \rightarrow \mathbb{Z}_q^n)$
2. $H' \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}} \rightarrow \{0,1\})$
3. $\hat{H} \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}} \rightarrow \{0,1\}^{\ell_c})$

Initialisation in $G_i$ for $i = 2, 3, 4, 5$:
1. $H' \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}} \rightarrow \{0,1\})$
2. $\hat{H} \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}} \rightarrow \{0,1\}^{\ell_c})$
3. $H'' \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}+\ell_{\mathsf{m}}} \rightarrow \{0,1\})$
4. $\tilde{H} \leftarrow U(\mathcal{F} : \{0,1\}^{\ell_{\mathsf{id}}+\ell_{\mathsf{m}}} \rightarrow \{0,1\}^{\ell_{c'}})$
```

Fig. 5. Functions for implementing quantum random oracles in G_i

SEUF-ID-CMA security experiment (Fig. 1). In the QROM, cryptographic hash functions H_1 and H_2 are initialised as truly random functions and maintained by the reduction. By our hypothesis, we have

$$\mathsf{Adv}^{\mathsf{seuf}}_{\Sigma',\mathcal{A}}(\lambda) := \Pr[\mathsf{Exp}^{\mathsf{seuf}}_{\Sigma',\mathcal{A}}(\lambda) = 1] = \Pr[S_0] \tag{2}$$

Game G_1 changes how H_1 is simulated, as shown in Fig. 5 and Fig. 7. In particular, in G_1 a random function, $\hat{H}$ is selected to produce the randomness for the algorithm $\mathsf{SampleR}$. A random function H' is selected to help simulate H_1. According to the specification of H_1 for G_1 from Fig. 7, we have

$$H_1(\mathsf{id}||b) := \begin{cases} \mathbf{A}\mathsf{SampleR}(m, w, s; \hat{H}(\mathsf{id})||b) + \mathbf{G}, & \text{if } b = H'(\mathsf{id}) \\ \mathbf{A}\mathsf{SampleR}(m, w, s; \hat{H}(\mathsf{id})||b), & \text{otherwise} \end{cases}$$

Using Lemma 10 with the facts that matrix $\mathbf{A}$'s column dimension $m \geq 2n \log q + \omega(\sqrt{\log n})$ and parameter $s \geq \omega(\sqrt{\log n})$, $\mathbf{A}\mathsf{SampleR}(m, w, s; \hat{H}(\mathsf{id})||b)$, and thus, $H_1(\mathsf{id}||b)$ are distributed $\mathsf{negl}'_1(\lambda)$-close to $U(\mathbb{Z}_q^{n \times w})$. Since H_1 is queried Q_{H_1} times. Using Lemma 1, we have for some negligible function $\mathsf{negl}_1(\lambda)$

```
$\mathsf{Setup}(1^\lambda)$ in $G_i$ for $i = 0, 1, 2, 3, 4$:
1. $(\mathbf{A}, \mathbf{R}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$
2. $L_{\mathsf{Sig}}, L_{\mathsf{Sk}} \leftarrow \emptyset$
3. Return $\mathsf{Mpk} := \mathbf{A}$, $\mathsf{Msk} := \{\mathbf{R}, k_1, k_2\}$

$\mathsf{Setup}(1^\lambda)$ in $G_5$:
1. $\mathbf{A} \leftarrow U(\mathbb{Z}_q^{n \times m})$
2. $L_{\mathsf{Sig}}, L_{\mathsf{Sk}} \leftarrow \emptyset$
3. Return $\mathsf{Mpk} := \mathbf{A}$, $\mathsf{Msk} := \emptyset$
```

Fig. 6. Setup for the G_i

Oracle $|H_1\rangle(\cdot)$ in G_0,:

1. On input superposition $\sum_{\mathsf{id},b,y} \alpha_{\mathsf{id},b,y} |\mathsf{id}||b, y\rangle$, return

$$\sum\nolimits_{\mathsf{id},b,y} \alpha_{\mathsf{id},b,y} |\mathsf{id}||b, H_1(\mathsf{id}||b) \oplus y\rangle$$

Oracle $|H_1\rangle(\cdot)$ in G_i for $i = 1, 2, 3, 4, 5$:

Set $\mathbf{R}_{\mathsf{id}||b} \leftarrow \mathsf{SampleR}(m, w, s; \hat{H}(\mathsf{id}||b))$ and define H_1 as:

$$H_1(\mathsf{id}||b) := \begin{cases} \mathbf{A}\mathbf{R}_{\mathsf{id}||b} + \mathbf{G}, & \text{if } b = H'(\mathsf{id}) \\ \mathbf{A}\mathbf{R}_{\mathsf{id}||b}, & \text{otherwise} \end{cases}$$

1. On input superposition $\sum_{\mathsf{id},b,y}, \alpha_{\mathsf{id},b,y} |\mathsf{id}||b, y\rangle$, return

$$\sum\nolimits_{\mathsf{id},b,y} \alpha_{\mathsf{id},b,y} |\mathsf{id}||b, H_1(\mathsf{id}||b) \oplus y\rangle$$

Fig. 7. $|H_1\rangle$ for game G_i

Oracle $|H_2\rangle(\cdot)$ in G_i for $i = 0, 1$:

1. On input superposition $\sum_{\mathsf{id},\mathsf{m},y} \alpha_{\mathsf{id},\mathsf{m},y} |\mathsf{id}||\mathsf{m}, y\rangle$, return

$$\sum\nolimits_{\mathsf{id},\mathsf{m},y} \alpha_{\mathsf{id},\mathsf{m},y} |\mathsf{id}||\mathsf{m}, H_2(\mathsf{id}||\mathsf{m}) \oplus y\rangle$$

Oracle $|H_2\rangle(\cdot)$ in G_i for $i = 2, 3, 4, 5$:

Set $\delta \leftarrow H''(\mathsf{id}||\mathsf{m})$ and define H_2 as:

$$H_2(\mathsf{id}||\mathsf{m}) := [\mathbf{A}|H_1(\mathsf{id}||\delta)]\mathsf{SampleR}(m + w, 1, s'; \tilde{H}(\mathsf{id}||\mathsf{m}))$$

1. On input superposition $\sum_{\mathsf{id},\mathsf{m},y} \alpha_{\mathsf{id},\mathsf{m},y} |\mathsf{id}||\mathsf{m}, y\rangle$, return

$$\sum\nolimits_{\mathsf{id},\mathsf{m},y} \alpha_{\mathsf{id},\mathsf{m},y} |\mathsf{id}||\mathsf{m}, H_2(\mathsf{id}||\mathsf{m}) \oplus y\rangle$$

Fig. 8. $|H_2\rangle$ for game G_i

$$|\Pr[S_1] - \Pr[S_0]| \leq 4Q_{H_1}^2 \sqrt{\mathsf{negl}_1'(\lambda)} \leq \mathsf{negl}_1(\lambda) \tag{3}$$

G_2 changes how H_2 is simulated. Two random functions are initialised, as shown in Fig. 5. $H'' : \{0,1\}^{\ell_{\mathsf{id}}+\ell_{\mathsf{m}}} \rightarrow \{0,1\})$ is used to specify the bit value that appears in the signature of $(\mathsf{id}, \mathsf{m})$. $\tilde{H} : \{0,1\}^{\ell_{\mathsf{id}}+\ell_{\mathsf{m}}} \rightarrow \{0,1\}^{\ell_{c'}}$ maps bit strings of length $\ell_{\mathsf{id}} + \ell_{\mathsf{m}}$ into $\ell_{c'}$-bit randomnesses for $\mathsf{SampleR}$. According to Fig. 8,

$$H_2(\mathsf{id}||\mathsf{m}) := [\mathbf{A}|H_1(\mathsf{id}||\delta)]\mathsf{SampleR}(m + w, 1, s'; \tilde{H}(\mathsf{id}||\mathsf{m}))$$

where $\delta \leftarrow H''(\mathsf{id}||\mathsf{m})$. Since H_1 is properly simulated (i.e., its output distribution is statistically close to $U(\mathbb{Z}_q^{n\times m})$), and $\mathsf{SampleR}(m+w, 1, s'; \tilde{H}(\mathsf{id}||\mathsf{m}))$ distributes statistically close to $D_{\mathbb{Z}^m,s'}$ where $s' \geq \omega(\sqrt{\log n})$, by Lemma 10, $H_2(\mathsf{id}||\mathsf{m})$ is distributed $\mathsf{negl}_2'(\lambda)$-close to $U(\mathbb{Z}_q^n)$. Using Lemma 1, for some negligible $\mathsf{negl}_2(\lambda)$

$$|\Pr[S_2] - \Pr[S_1]| \leq 4Q_{H_2}^2 \sqrt{\mathsf{negl}_2'(\lambda)} \leq \mathsf{negl}_2(\lambda) \tag{4}$$

```
OSig(id, m) in $G_i$ for $i = 0, 1, 2$:
Return $\sigma$ if $(\mathsf{id}, \mathsf{m}, \sigma) \in L_{\mathsf{Sig}}$
$\mathsf{Sk}_{\mathsf{id}||b} \leftarrow \mathsf{OExt}(\mathsf{id}) := \mathsf{Sk}_{\mathsf{id}} = (\mathbf{R}_{\mathsf{id}||b}, b)$
$\mathbf{u} \leftarrow H_2(\mathsf{id}||\mathsf{m})$
$\mathbf{A}_{\mathsf{id}||b} \leftarrow H_1(\mathsf{id}||b)$, $\mathbf{F}_{\mathsf{id}||b} := [\mathbf{A}|\mathbf{A}_{\mathsf{id}||b}]$
$\varphi \leftarrow U(\{0,1\}^{\ell_{s'}})$
$\mathbf{d} \leftarrow \mathsf{SampleD}(\mathbf{F}_{\mathsf{id}||b}, \mathbf{R}_{\mathsf{id}||b}, \mathbf{u}, s'; \varphi)$
$L_{\mathsf{Sig}} \leftarrow \{(\mathsf{id}, \mathsf{m}, \sigma)\} \cup L_{\mathsf{Sig}}$
Return $\sigma := (\sigma_1, \sigma_2) := (\mathbf{d}, b)$
$L_\sigma \leftarrow \{(\mathsf{id}, \mathsf{m}, \sigma)\} \cup L_\sigma$
```

```
OSig(id, m) in $G_i$ for $i = 3, 4, 5$:
Return $\sigma$ if $(\mathsf{id}, \mathsf{m}, \sigma) \in L_{\mathsf{Sig}}$
$\delta \leftarrow H''(\mathsf{id}||\mathsf{m})$
$\mathbf{d} \leftarrow \mathsf{SampleR}(m + w, 1, s; \tilde{H}(\mathsf{id}||\mathsf{m}))$
$L_{\mathsf{Sig}} \leftarrow \{(\mathsf{id}, \mathsf{m}, \sigma)\} \cup L_{\mathsf{Sig}}$
Return $\sigma := (\sigma_1, \sigma_2) = (\mathbf{d}, \delta)$
$L_\sigma \leftarrow \{(\mathsf{id}, \mathsf{m}, \sigma)\} \cup L_\sigma$
```

Fig. 9. Oracle $\mathsf{OSig}()$ for game G_i

```
OExt(id) in $G_i$ for $i = 0, 1, 2, 3$:
Return $\mathsf{Sk}_{\mathsf{id}}$ if $(\mathsf{id}, \mathsf{Sk}_{\mathsf{id}}) \in L_{\mathsf{Sk}}$
$b \leftarrow U(\{0,1\})$, $\psi \leftarrow U(\{0,1\}^{\ell_s})$
$\mathbf{A}_{\mathsf{id}||b} \leftarrow H_1(\mathsf{id}||b)$
$\mathbf{R}_{\mathsf{id}||b} \leftarrow \mathsf{DelTrap}(\mathbf{A}, \mathbf{R}, \mathbf{A}_{\mathsf{id}||b}, s; \psi)$
Return $\mathsf{Sk}_{\mathsf{id}} := (\mathbf{R}_{\mathsf{id}||b}, b)$
$L_{\mathsf{Sk}} \leftarrow \{(\mathsf{id}, \mathsf{Sk}_{\mathsf{id}})\} \cup L_{\mathsf{Sk}}$
$L_{\mathsf{id}} \leftarrow \{\mathsf{id}\} \cup L_{\mathsf{id}}$
```

```
OExt(id) in $G_4$, $G_5$:
Return $\mathsf{Sk}_{\mathsf{id}}$ if $(\mathsf{id}, \mathsf{Sk}_{\mathsf{id}}) \in L_{\mathsf{Sk}}$
$b \leftarrow H'(\mathsf{id})$
$\mathbf{R}_{\mathsf{id}||b} \leftarrow \mathsf{SampleR}(m, w, s; \hat{H}(\mathsf{id}||b))$
Return $\mathsf{Sk}_{\mathsf{id}} := (\mathbf{R}_{\mathsf{id}||b}, b)$
$L_{\mathsf{Sk}} \leftarrow \{(\mathsf{id}, \mathsf{Sk}_{\mathsf{id}})\} \cup L_{\mathsf{Sk}}$
$L_{\mathsf{id}} \leftarrow \{\mathsf{id}\} \cup L_{\mathsf{id}}$
```

Fig. 10. Oracle $\mathsf{OExt}()$ for game G_i

G_3 changes OSig, as specified in Fig. 9[6]. Assume the signature of $(\mathsf{id}, \mathsf{m})$ is $\sigma := (\mathbf{d}, b)$. In G_2, b is uniformly random. In G_3, $\delta \leftarrow H''(\mathsf{id}||\mathsf{m})$ for random function H''. So, b is distributed identically to δ. In G_2, according to Lemma 8, the vector $\mathbf{d} \leftarrow \mathsf{SampleD}([\mathbf{A}|\mathbf{A}_{\mathsf{id}||b}], \mathbf{R}_{\mathsf{id}||b}, H_2(\mathsf{id}||\mathsf{m}), s')$ has a distribution statistically close to $D_{\mathbb{Z}^{m+w},s'}$ conditioned on $[\mathbf{A}|H_1(\mathsf{id}||b)]\mathbf{d} = H_2(\mathsf{id}||\mathsf{m})$. In G_3, $\mathbf{d} \leftarrow \mathsf{SampleR}(m + w, 1, s; \tilde{H}(\mathsf{id}||\mathsf{m}))$. By Lemma 10, $\mathbf{d}$ has a distribution statistically close to $D_{\mathbb{Z}^{m+w},s'}$. Also, $H_2(\mathsf{id}||\mathsf{m}) := [\mathbf{A}|H_1(\mathsf{id}||\delta)]\mathbf{d}$ where δ is uniformly random, as b. So, the distribution of $(\mathbf{d}, b)$ in G_2 and the distribution of $(\mathbf{d}, \delta)$ are negligibly close. So, for some negligible $\mathsf{negl}_3(\lambda)$

$$|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{negl}_3(\lambda) \tag{5}$$

G_4 changes how OExt works, as specified in Fig. 10. Recall that for OExt in G_3, the bit value b is uniformly random. For OExt in G_4, $b \leftarrow H'(\mathsf{id})$. Since H' is truly random and not accessible to the adversary, the distributions of b in G_3, G_4 are identical. Moreover, in G_3, for fixed $\mathbf{A}_{\mathsf{id}||b}$, $\mathbf{R}_{\mathsf{id}||b}$ is sampled according to $\mathsf{DelTrap}(\mathbf{A}, \mathbf{R}, \mathbf{A}_{\mathsf{id}||b}, s)$, i.e., $\mathbf{R}_{\mathsf{id}||b}$ has a distribution that is statistically close to $D^{w}_{\mathbb{Z}^m,s}$, conditioned on $\mathbf{A}_{\mathsf{id}||b} = \mathbf{A}\mathbf{R}_{\mathsf{id}||b} + \mathbf{G}$. As per Lemma 9, $\mathbf{R}_{\mathsf{id}||b}$ has a distribution statistically close to $D_{\Lambda_q^{\mathbf{A}_{\mathsf{id}||b}-\mathbf{G}}(\mathbf{A}),s}$. In G_4, $\mathbf{R}_{\mathsf{id}||b} \leftarrow \mathsf{SampleR}(m, w, s; \hat{H}(\mathsf{id}||b)$ which distributes statistically close to $D^{w}_{\mathbb{Z}^m,s}$. Meanwhile, according to the simulation of H_1, $H_1(\mathsf{id}||b) = H_1(\mathsf{id}||H'(\mathsf{id})) :=$

[6] We note that in Fig. 9 the lists L_{Sig} and L_σ are identical, both storing signed messages, the corresponding identities and signatures. L_{Sig} is part of the signing algorithm (to make signing stateful), and the security games maintain L_σ.

$\mathbf{A}_{\mathsf{id}||b} = \mathbf{A}\mathbf{R}_{\mathsf{id}||b} + \mathbf{G}$. As per Lemma 10, the distributions of $\mathbf{R}_{\mathsf{id}||b}$ in G_3 and G_4 are $\mathsf{negl}_4(\lambda)$-close for some $\mathsf{negl}_4(\lambda)$.

$$|\Pr[S_4] - \Pr[S_3]| \leq \mathsf{negl}_4(\lambda) \tag{6}$$

G_5 changes how Setup works, as showed by Fig. 6. In particular, the public matrix $\mathbf{A} \leftarrow U(\mathbb{Z}_q^{n\times m})$ and Msk is set empty. By Lemma 7, $\mathcal{A}$'s distributions in G_4 and G_5 are $\mathsf{negl}_5(\lambda)$-close for some negligible $\mathsf{negl}_5(\lambda)$. Note that the adversary $\mathcal{A}$ is not given Msk in G_4 and G_5, we conclude that

$$|\Pr[S_5] - \Pr[S_4]| \leq \mathsf{negl}_5(\lambda) \tag{7}$$

We construct an efficient algorithm $\mathcal{B}$ that solves the SIS problem. $\mathcal{B}$ receives a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ which is a $\mathsf{SIS}_{n,q,m,\beta}$ problem instance. $\mathcal{B}$ runs G_5 with the adversary $\mathcal{A}$ (the algorithm maintains all procedures and random oracles as per the specification of G_5). If G_5 outputs 0, $\mathcal{B}$ aborts. If G_5 outputs 1, i.e., the event S_5 happens, $\mathcal{A}$ must have outputted $(\mathsf{id}^*, \mathsf{m}^*, \sigma^*) \notin L_\sigma$ (i.e., σ^* is not produced by $\mathsf{OSig}(\mathsf{id}^*, \mathsf{m}^*)$) such that $\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}^*, \mathsf{m}^*, \sigma^*) = 1$ where $\mathsf{id}^* \notin L_{\mathsf{id}}$, assume $\sigma^* := (\sigma_1^*, \sigma_2^*) = (\mathbf{d}^* \in \mathbb{Z}^{m+w}, b^* \in \{0,1\})$ and $\mathbf{d}^* = \begin{bmatrix}\mathbf{d}_1^* \\ \mathbf{d}_2^*\end{bmatrix}$ with $\mathbf{d}_1^* \in \mathbb{Z}^m$ and $\mathbf{d}_2^* \in \mathbb{Z}^w$. $\mathcal{B}$ does:

1. Compute $\mu^* \leftarrow H'(\mathsf{id}^*)$, $\delta^* \leftarrow H''(\mathsf{id}^*||\mathsf{m}^*)$, and abort if $b^* \neq 1 - \mu^*$ or $\delta^* \neq 1 - \mu^*$; Otherwise
2. Return $\mathbf{e} = [\mathbf{I}_m|\mathbf{R}^*](\mathbf{d} - \mathbf{d}^*)$ where $\mathbf{R}^* = \mathsf{SampleR}(m, w, s; \hat{H}(\mathsf{id}^*||H'(\mathsf{id}^*)))$ as the SIS solution.

The following three lemmas demonstrate that with a probability close to 1/4, $\mathbf{e}$ is a valid SIS solution, i.e., $\mathbf{A}\mathbf{e} = \mathbf{0}$, $\mathbf{e} \neq \mathbf{0}$ and $\|\mathbf{e}\| \leq \beta$.

Lemma 11. *The event $b^* = 1 - \mu^* = \delta^*$ happens with probability* 1/4.

Proof. In G_5, the only oracles that may give $\mathcal{A}$ the information about $H'(\mathsf{id}^*)$ are H_1 and OExt. As shown in the proof of the inequality 3, the output distribution of H_1 is statistically close to $U(\mathbb{Z}_q^{n\times w})$. So, H_1 leaks no information about $H'(\mathsf{id}^*)$. Moreover, id^* is not allowed to query OExt as per the SEUF-ID-CMA definition. So, $\mu^* = H'(\mathsf{id}^*)$ remains random from $\mathcal{A}$'s view. Similarly, the only oracles may leak information about $\delta^* = H''(\mathsf{id}^*||\mathsf{m}^*)$ are H_2 and OSig. However, we have proved that the output distribution of H_2 is statistically close to $U(\mathbb{Z}_q^n)$, and bears no information about $H''(\mathsf{id}^*||\mathsf{m}^*)$. Also, $(\mathsf{id}^*, \mathsf{m}^*)$ is not allowed to query OSig as per the definition of SEUF-ID-CMA. Hence, δ^* remains random from $\mathcal{A}$'s view, and, $1 - b^* = \mu^* = 1 - \delta^*$ happens with probability 1/4. □

Lemma 12. *Provided the condition $b^* = 1 - \mu^* = \delta^*$ holds,* $\mathbf{A}\mathbf{e} = \mathbf{0}$.

Proof. As the forged signature $\sigma^* = (\mathbf{d}^*, b^*)$ is valid, we must have

$$[\mathbf{A}|H_1(\mathsf{id}^*||b^*)]\mathbf{d}^* = H_2(\mathsf{id}^*||\mathsf{m}^*).$$

According to the simulation of H_2 in G_5 (i.e., Fig. 8),

$$[\mathbf{A}|H_1(\mathsf{id}^*||\delta^*)]\mathbf{d} = H_2(\mathsf{id}^*||\mathsf{m}^*)$$

where $\delta^* \leftarrow H''(\mathsf{id}^*||\mathsf{m}^*)$ and $\mathbf{d} = \mathsf{SampleR}(m+w, 1, s'; \tilde{H}(\mathsf{id}^*||\mathsf{m}^*))$. If $b^* = 1-\mu^* = \delta^*$, according to the simulation of H_1 in G_5 (i.e., Fig. 7), $H_1(\mathsf{id}^*||b^*) = H_1(\mathsf{id}^*||\delta^*) = \mathbf{AR}^*$ where $\mathbf{R}^* = \mathsf{SampleR}(m, w, s; \hat{H}(\mathsf{id}^*||1-\mu^*))$. Consequently, we obtain $[\mathbf{A}|\mathbf{AR}^*](\mathbf{d}-\mathbf{d}^*) = \mathbf{0}$ and thus $\mathbf{Ae} = \mathbf{0}$. □

Lemma 13. *Let* $\mathbf{e}$ *be constructed as above. We have* $\mathbf{e} \neq \mathbf{0}$ *and* $\|\mathbf{e}\| \leq \beta$ *where* $\beta = O(w^{3/2}) \cdot \omega(\sqrt{\log n})^2$ *with all but negligible probability.*

Proof. To show that $\|\mathbf{e}\| \leq \beta$, we have

$$\begin{aligned}\|\mathbf{e}\| &\leq \|(\mathbf{d}_1 - \mathbf{d}_1^*)\| + s_1(\mathbf{R}_{\mathsf{id}^*||b^*}) \cdot \|\mathbf{d}_2 - \mathbf{d}_2^*\| \\ &\leq 2s'\sqrt{m} + (C \cdot s(\sqrt{m} + \sqrt{w})) \cdot 2s'\sqrt{w} \\ &\leq O(w^{3/2}) \cdot \omega(\sqrt{\log n})^2 = \beta\end{aligned}$$

To show $\mathbf{e} \neq \mathbf{0}$ w.h.p. Let $\mathbf{d} = \begin{bmatrix}\mathbf{d}_1\\ \mathbf{d}_2\end{bmatrix}$ where $\mathbf{d}_1 \in \mathbb{Z}^m$, $\mathbf{b}_2 \in \mathbb{Z}^w$. We consider the following cases:

- $\mathbf{x} = \mathbf{d}_2 - \mathbf{d}_2^* = \mathbf{0}$: In this case, $\mathbf{y} = \mathbf{d}_1 - \mathbf{d}_1^* \neq \mathbf{0}$. If $\mathcal{A}$ has queried $(\mathsf{id}^*, \mathsf{m}^*)$ to OSig and received $(\mathbf{d}, \delta^*) = (\mathbf{d}, b^*)$ as the answer (note this is consistent with the simulation of OSig), we must have $\mathbf{d} \neq \mathbf{d}^*$ and thus, $\mathbf{e} \neq \mathbf{0}$.
- $\mathbf{y} = \mathbf{d}_1 - \mathbf{d}_1^* \neq \mathbf{0}$: In this case, $\mathbf{y}$ must have a non-zero coordinate. W.l.o.g, assume last coordinate of $\mathbf{y}$, y is non-zero. Then, $\mathbf{e}$ can be written as $\mathbf{e} = \mathbf{x} + \mathbf{r}^* y + \mathbf{v}$ where $\mathbf{r}^*$ is the last column of $\mathbf{R}^*$, $\mathbf{v} \in \mathbb{Z}^m$ depends on $\mathbf{R}^*$'s remaining columns and $\mathbf{y}$'s remaining coordinates. Note, $\mathbf{R}^*$ (also $\mathbf{r}^*$) is not given to the adversary $\mathcal{A}$. The only information about $\mathbf{r}^*$ that $\mathcal{A}$ can get is via $H_1(\mathsf{id}^*||b^*) = \mathbf{AR}^*$. By Lemma 10, $H_1(\mathsf{id}^*||b^*)$ is statistically close to $U(\mathbb{Z}_q^{n\times w})$ w.h.p. So, conditioned on $H_1(\mathsf{id}^*||b^*) = \mathbf{AR}^*$, $\mathbf{r}^*$ is distributed negligibly close to $D_{\mathbb{Z}^m,s}$. By Lemma 5, $\mathbf{r}^*$ has min-entropy $\geq m-1$. So, $\mathbf{e} \neq \mathbf{0}$ with probability $1 - 2^{-(m-1)} = 1 - \mathsf{negl}_6'(\lambda)$ for some negligible $\mathsf{negl}_6'(\lambda)$. □

Let E be the event that the algorithm $\mathcal{B}$ solves the SIS problem challenge. Combining Lemma 12, Lemma 13 and Lemma 11 leads to

$$\begin{aligned}\Pr[E] &= \Pr[E|S_5]\Pr[S_5] + \Pr[E|\neg S_5]\Pr[\neg S_5] \\ &\geq \Pr[E|S_5]\Pr[S_5] = \frac{1}{4}(1 - \mathsf{negl}_6'(\lambda))\Pr[S_5] \\ &= \frac{1}{4}\Pr[S_5] - \mathsf{negl}_6'(\lambda) \cdot \Pr[S_5]\end{aligned}$$

Note that $\Pr[E] \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SIS}_{n,q,m,\beta}}(\lambda)$. Thus,

$$\Pr[S_5] \leq 4 \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{SIS}_{n,q,m,\beta}}(\lambda) + \mathsf{negl}_6(\lambda) \quad (8)$$

for negligible $\mathsf{negl}_6(\lambda) = 4 \cdot \mathsf{negl}'_6(\lambda) \cdot \Pr[S_5]$. Putting the inequalities 2, 3, 4, 5, 6, 7, and 8 together, we have

$$\mathsf{Adv}^{\mathsf{seuf}}_{\Sigma',\mathcal{A}}(\lambda) \leq 4 \cdot \mathsf{Adv}^{\mathsf{SIS}_{n,q,m,\beta}}_{\mathcal{B}}(\lambda) + \mathsf{negl}'(\lambda)$$

where $\mathsf{negl}'(\lambda)$ upper bounds the negligible terms $\sum_{i=1}^{6} \mathsf{negl}_i(\lambda)$. This concludes the SEUF-ID-CMA security proof for the scheme Σ'. □

4.3 The Security of IBS Scheme $\boldsymbol{\Sigma}$

Proof. Combining the proofs from Sect. 4.1 and Sect. 4.2 results in

$$\begin{aligned}\mathsf{Adv}^{\mathsf{seuf}}_{\Sigma,\mathcal{A}}(\lambda) &\leq 2(Q_{\mathsf{Ext}} + Q_{\mathsf{Sig}}) \cdot 2^{-\ell/2} + \mathsf{Adv}^{\mathsf{seuf}}_{\Sigma',\mathcal{A}}(\lambda) \\ &\leq 4 \cdot \mathsf{Adv}^{\mathsf{SIS}_{n,q,m,\beta}}_{\mathcal{B}}(\lambda) + \mathsf{negl}(\lambda)\end{aligned}$$

where the negligible $\mathsf{negl}(\lambda)$ upper bounds $\mathsf{negl}'(\lambda) + 2(Q_{\mathsf{Ext}} + Q_{\mathsf{Sig}}) \cdot 2^{-\ell/2}$. Let the running time of the attacker $\mathcal{A}$ and the time used by simulations in the proofs be t' and t. t is bounded by $t' + (Q_{H_1} + Q_{H_2}) \cdot \mathsf{poly}(\lambda)$ as proof simulations run $\mathcal{A}$ as a subroutine and the extra computations of the simulation can be done in polynomial time. □

5 Security Proof (Sketch) in the Random Oracle Model

Theorem 2. *Let H_1, H_2, H_1, H_2, H_3 be random oracles. If (t, ϵ)-$\mathsf{SIS}_{n,q,m,\beta}$ assumption holds with $\beta = O(w^{3/2})\omega(\sqrt{\log n})^2$, the IBS scheme Σ is (t', Q_{Ext}, Q_{Sig}, $4\epsilon + \mathsf{negl}(\lambda)$)-SEUF-ID-CMA secure for $t' \leq t + O(Q_{H_1} + Q_{H_2}) \cdot \mathsf{poly}(\lambda)$ where Q_{H_1} and Q_{H_2} are the numbers of random oracle queries to $|H_1\rangle$ and $|H_2\rangle$.*

Proof (Sketch). The theorem states the security of our IBS scheme in the classic ROM. We sketch the security proof due to the similarity to the QROM proof. Similar to the QROM proof, we first reduce the SEUF-ID-CMA security of the IBS scheme Σ (Fig. 2) to the stateful IBS scheme Σ' defined in Fig. 3. This step replaces the random oracles H_1, H_2, H_3 by internal states, and uses Lemma 2 to get inequality 1, showing the security gap between Σ and Σ' is negligible.

Then, we construct a tight reduction $\mathcal{B}$ in Fig. 11 that converts the adversary $\mathcal{A}$'s advantage agaisnt the stateful IBS scheme Σ' to its advantage in solving the SIS problem. On receiving the SIS challenge $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathcal{B}$ runs the sub-routines $\mathsf{Setup}_{\mathcal{B}}$, $\mathsf{Sig}_{\mathcal{B}}$, $\mathsf{Ext}_{\mathcal{B}}$, and maintains the random oracles H_1, H_2 to simulate the IBS scheme Σ' perfectly. Finally, it uses the sub-routine $\mathsf{Sol}_{\mathcal{B}}$ to extract the SIS problem solution $\mathbf{e}$. Like the proof in the QROM, the factor 4 in the security loss comes from requiring the condition $\mu^* = 1 - b^* = 1 - \delta^*$ holds.

Finally, similar to the QROM proof, the running time of the first-step reduction is nearly the same as the corresponding adversary, and the running time of the reduction $\mathcal{B}$'s running time t is bounded by $t' + (Q_{H_1} + Q_{H_2}) \cdot \mathsf{poly}(\lambda)$ where t' is the running time against the statful IBS scheme Σ'. □

```
The algorithm $\mathcal{B}(\mathbf{A})$:
Input: $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$
Goal: Find $\mathbf{e} \neq \mathbf{0}$ s.t. $\mathbf{Ae} = \mathbf{0} \bmod q$, $\|\mathbf{e}\| \leq \beta$
1. $L_1, L_2, L_{\mathsf{id}}, L_\sigma \leftarrow \emptyset$
2. $(\mathsf{Mpk}, \mathsf{Msk}) \leftarrow \mathsf{Setup}_{\mathcal{B}}(1^\lambda)$
3. $(\mathsf{id}^*, \mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Ext}_{\mathcal{B}}(), \mathsf{Sig}_{\mathcal{B}}(), H_1, H_2}(\mathsf{Mpk}, \mathsf{st})$
4. If $\mathsf{id}^* \in L_{\mathsf{id}}$ or $(\mathsf{id}^*, \mathsf{m}^*, \sigma^*) \in L_\sigma$, return $\perp$
5. If $\mathsf{Ver}(\mathsf{Mpk}, \mathsf{id}^*, \mathsf{m}^*, \sigma^*) = 0$, return $\perp$
6. Return $\mathsf{Sol}_{\mathcal{B}}(\mathsf{id}^*, \mathsf{m}^*, \sigma^*)$

Sub-routine $\mathsf{Setup}_{\mathcal{B}}(1^\lambda)$
1. $L_{\mathsf{Sig}}, L_{\mathsf{Sk}} \leftarrow \emptyset$ [a]
2. Return $\mathsf{Mpk} := \mathbf{A}$, $\mathsf{Msk} := \emptyset$

Random oracle $H_1(\mathsf{id}||b)$
1. If $(\mathsf{id}||b, \mu, \mathbf{R}_{\mathsf{id}||b}, \mathbf{A}_{\mathsf{id}||b}) \in L_1$, return $\mathbf{A}_{\mathsf{id}||b}$
2. $\mu \leftarrow U(\{0,1\})$, $\mathbf{R}_{\mathsf{id}||b}, \mathbf{R}_{\mathsf{id}||1-b} \leftarrow D_{\mathbb{Z},s}^{m \times w}$
3. $\mathbf{A}_{\mathsf{id}||\mu} \leftarrow \mathbf{AR}_{\mathsf{id}||\mu} + \mathbf{G}$, $\mathbf{A}_{\mathsf{id}||1-\mu} \leftarrow \mathbf{AR}_{\mathsf{id}||1-\mu}$ [b]
4. $L_1 \leftarrow L_1 \cup \{(\mathsf{id}||b, \mu, \mathbf{R}_{\mathsf{id}||b}, \mathbf{A}_{\mathsf{id}||b}), (\mathsf{id}||1-b, \mu, \mathbf{R}_{\mathsf{id}||1-b}, \mathbf{A}_{\mathsf{id}||1-b})\}$.
5. Return $\mathbf{A}_{\mathsf{id}||b} \in \mathbb{Z}_q^{m \times w}$

Random oracle $H_2(\mathsf{id}||\mathsf{m})$
1. If $(\mathsf{id}||\mathsf{m}, b, \mathbf{d}, \mathbf{u}) \in L_2$, return $\mathbf{u}$
2. $\delta \leftarrow U(\{0,1\})$, $\mathbf{A}_{\mathsf{id}||\delta} \leftarrow H_1(\mathsf{id}||\delta)$
3. $\mathbf{d} \leftarrow D_{\mathbb{Z},s'}^{m+w}$, $\mathbf{u} \leftarrow [\mathbf{A}|\mathbf{A}_{\mathsf{id}||\delta}]\mathbf{d}$
4. $L_2 \leftarrow L_2 \cup \{(\mathsf{id}||\mathsf{m}, \delta, \mathbf{d}, \mathbf{u})\}$

Sub-routine $\mathsf{Sig}_{\mathcal{B}}(\mathsf{id}, \mathsf{m})$
1. Return $\sigma$ if $(\mathsf{id}, \mathsf{m}, \sigma) \in L_{\mathsf{Sig}}$
2. $\mathbf{u} \leftarrow H_2(\mathsf{id}||\mathsf{m})$
3. Identify $(\mathsf{id}||\mathsf{m}, b, \mathbf{d}, \mathbf{u}) \in L_2$
4. $L_{\mathsf{Sig}} \leftarrow \{(\mathsf{id}, \mathsf{m}, \sigma)\} \cup L_{\mathsf{Sig}}$
5. Return $\sigma := (\mathbf{d}, b)$
6. $L_\sigma \leftarrow L_\sigma \cup \{(\mathsf{id}, \mathsf{m}, \sigma)\}$

Sub-routine $\mathsf{Ext}_{\mathcal{B}}(\mathsf{id})$
1. Return $\mathsf{Sk}_{\mathsf{id}}$ if $(\mathsf{id}, \mathsf{Sk}_{\mathsf{id}}) \in L_{\mathsf{Sk}}$
2. $b \leftarrow U(\{0,1\})$
3. $\mathbf{A}_{\mathsf{id}||b} \leftarrow H_1(\mathsf{id}||b)$
4. Identify $(\mathsf{id}||\mu, \mu, \mathbf{R}_{\mathsf{id}||b}, \mathbf{A}_{\mathsf{id}||b}) \in L_1$
5. Return $\mathsf{Sk}_{\mathsf{id}} := (\mathbf{R}_{\mathsf{id}||\mu}, \mu)$
6. $L_{\mathsf{Sk}} \leftarrow \{(\mathsf{id}, \mathsf{Sk}_{\mathsf{id}})\} \cup L_{\mathsf{Sk}}$
7. $L_{\mathsf{id}} \leftarrow L_{\mathsf{id}} \cup \{\mathsf{id}\}$

Sub-routine $\mathsf{Sol}_{\mathcal{B}}(\mathsf{id}^*, \mathsf{m}^*, \sigma^*)$
1. $\sigma^* := (\mathbf{d}^*, b^*)$
2. Identify $(\mathsf{id}^*||\mathsf{m}^*, \delta^*, \mathbf{d}, \mathbf{u}) \in L_2$,
   $(\mathsf{id}^*||b^*, \mu^*, \mathbf{R}^*_{\mathsf{id}||b}, \mathbf{A}_{\mathsf{id}^*||b^*}) \in L_1$
3. Return $\perp$ if $b^* = \mu^*$ or $\delta^* = \mu^*$;
4. Return $\mathbf{e} \leftarrow [\mathbf{I}_m|\mathbf{R}^*_{\mathsf{id}^*||b^*}] \cdot (\mathbf{d} - \mathbf{d}^*)$
```

[a] Fig. 11, the lists L_{Sig} and L_σ are identical. L_{Sig} is part of the signing algorithm and L_σ is maintained by the security games.

[b] $\mathbf{G}$ is equipped by $\mathbf{A}_{\mathsf{id}||b}$ when $\mu = b$.

Fig. 11. Reduction to SIS in Random Oracle Model

6 Conclusion

We have presented a short lattice IBS scheme. We proved, in detail, that the scheme is strongly existentially unforgeable under the chosen-identity attack and the chosen-message attack based on the standard SIS problem with a tight security reduction in the QROM and ROM. Our scheme outperforms the existing schemes with shorter signatures and a stronger unforgeability and relying on weaker SIS assumptions. We leave constructing a short and tightly quantum-secure IBS scheme in the plain model as a future research question.

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_28
2. Blazy, O., Kakvi, S.A., Kiltz, E., Pan, J.: Tightly-secure signatures from chameleon hash functions. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 256–279. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_12
3. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3

4. Boyen, X., Li, Q.: Towards tightly secure lattice short signature and id-based encryption. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 404–434. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_14
5. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. J. Cryptol. **25**(4), 601–639 (2012)
6. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC 2008, pp. 197–206 ACM (2008)
7. Gjøsteen, K., Jager, T.: Practical and tightly-secure digital signatures and authenticated key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 95–125. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_4
8. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC 1996, pp. 212–219, ACM (1996)
9. Katsumata, S., Yamada, S., Yamakawa, T.: Tighter security proofs for GPV-IBE in the quantum random oracle model. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 253–282. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_9
10. Katz, J., Wang, N.: Efficiency improvements for signature schemes with tight security reductions. In: CCS 2003, pp. 155–164, ACM (2003)
11. Kuchta, V., Sakzad, A., Stehlé, D., Steinfeld, R., Sun, S.-F.: Measure-rewind-measure: tighter quantum random oracle model proofs for one-way to hiding and CCA security. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 703–728. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_24
12. Lee, Y., Park, J.H., Lee, K., Lee, D.H.: Tight security for the generic construction of identity-based signature (in the multi-instance setting). Theor. Comput. Sci. **847**, 122–133 (2020)
13. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
14. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput. **37**(1), 267–302 (2007)
15. Pan, J., Wagner, B.: Short identity-based signatures with tight security from lattices. In: Cheon, J.H., Tillich, J.-P. (eds.) PQCrypto 2021. LNCS, vol. 12841, pp. 360–379. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81293-5_19
16. Pan, J., Wagner, B.: Short identity-based signatures with tight security from lattices. Cryptology ePrint Archive, Report 2021/970 (2021). https://eprint.iacr.org/2021/970
17. Pan, J., Wagner, B.: Lattice-based signatures with tight adaptive corruptions and more. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography (PKC 2022). LNCS, vol. 13178, pp. 347–378. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97131-1_12
18. Peikert, C.: A decade of lattice cryptography. Found. Trends Theor. Comput. Sci. **10**(4), 283–424 (2016)

19. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 520–551. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_17
20. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_5

Ghidle: Efficient Large-State Block Ciphers for Post-quantum Security

Motoki Nakahashi[1], Rentaro Shiba[2](✉), Ravi Anand[1], Mostafizar Rahman[1],
Kosei Sakamoto[1], Fukang Liu[1], and Takanori Isobe[1]

[1] University of Hyogo, Kobe, Japan
takanori.isobe@ai.u-hyogo.ac.jp
[2] Mitsubishi Electric Corporation, Kamakura, Japan
shiba.rentaro@dc.mitsubishielectric.co.jp

Abstract. In this paper we propose a new family of highly efficient and quantum secure AES-based block cipher dubbed Ghidle, which supports a key size of 256 bits and a state size of 256 or 512 bits. The large state size implies a "bigger birthday bound" security when these are embedded in modes of operation. Ghidle achieves high efficiency in both the encryption and the decryption by taking advantage of three consecutive executions of AES rounds in AES-NI, while Pholkos, which is an existing quantum-secure block cipher, is designed to be fast for only encryption performance due to the limitation of two consecutive executions of AES rounds. We run benchmarks of Ghidle on x86(_64) and arm64 environments and compare their performance with Pholkos. In our performance evaluation on modern x86 processors, the decryption of Ghidle-512 outperforms that of Pholkos-512 by about 54% while the encryption performance remains the same. We also evaluate the performance on mobile devices with arm64-based processors and the result shows that Ghidle-256 outperforms Pholkos-256 by about 32% for decryption while the encryption remains almost the same. Furthermore, Ghidle-512 outperforms Pholkos-512 by about 21% and 53% for both encryption and decryption, respectively.

Keywords: AES-NI · wide-block encryption · post-quantum cryptography

1 Introduction

1.1 Background

Security Requirements for Block Ciphers. In recent years, a lot of effort has been devoted to the development and practical applications of quantum computers. Along with this, the impact of quantum computers on the security of cryptographic primitives is also being studied. Symmetric cryptography is not as significantly affected by quantum computers as asymmetric cryptography, but

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 403–430, 2023.
https://doi.org/10.1007/978-3-031-35486-1_18

some quantum algorithms can substantially reduce the complexity of the analysis. Indeed, the 3GPP standardization organization also recognizes the possible impacts of quantum computing on symmetric cryptography for 5G and beyond 5G.

For instance, Grover's algorithm [22] can be applied to exhaustively search the k-bit key of symmetric ciphers to quadratically speed up the search complexity from $O(2^k)$ to $O(2^{k/2})$. Doubling the key of existing symmetric ciphers is a useful heuristic, especially against Grover's search algorithm, but its practical impact remains to be understood better. Nevertheless, a key size of at least 256-bit is a minimum requirement for 128-bit security in quantum setting [16].

In addition, block size should also be carefully specified in order to obtain effective security against quantum adversaries. Modes of operation or authenticated encryptions based on a n-bit block cipher provide security up to birthday bound $O(2^{n/2})$. For block ciphers with smaller state sizes, such as 64 bits, this might lead to practical attacks. While 128-bit block ciphers will not be affected as of now, in the future with the increase in computing powers of the adversaries and the rise of quantum computers, even 128-bit block ciphers might not be safe. A similar argument has also been presented in [7] and a possible solution put forward was to design ciphers with at least 256-bit blocks and keys - heavyweight ciphers - which would then provide a "bigger birthday bound" security. Based on these facts, we believe block ciphers require a key as well as a state size of at least 256-bits.

Performance Requirements for Block Ciphers. Furthermore, the block ciphers that ensure post-quantum security must be designed to be used in the future when quantum computers become widespread. In such a future, computer hardware will have higher computing speeds and network communication systems like Beyond 5G/6G will have much lower latency, which may require block ciphers to have even lower latency than today.

Previous Attempts. As efficient quantum-secure block ciphers, Saturnin [15] and Pholkos [14] were proposed.

Saturnin is a lightweight block cipher for NIST lightweight cryptography competition. Saturnin is efficient in resource-constrained environments, but it is not suitable for the software implementation on environments with high-end processors such as x86(_64) and arm64 because of the absence of support for the nibble-wise operations underlying operations of the Saturnin round.

On the other hand, Pholkos [14] is dedicated to high-end processors such as x86 and arm64. Pholkos uses the AES round function and the SIMD-friendly shuffle operation. In modern x86 processors, the encryption of Pholkos is highly efficient due to the availability of SIMD and instructions dedicated to AES such as AES-NI instructions. However, the decryption is not efficient because more instructions than the encryption is required. Besides, in arm64, the encryption and the decryption have the same performance but the number of instructions required to implement both of them increases significantly compared to the x86 implementation.

1.2 Motivation

At present, no block cipher has been proposed that satisfies the following two requirements: (1) Quantum security, namely at least 256-bit key and 256-bits block size. (2) High performance on software for both encryption and decryption performance. Our aim in this paper is to propose block ciphers that satisfy both requirements. Pholkos and Saturnin satisfy the first requirement, but not the second.

In [1,31], it is shown that some modes of operation such as CBC and XTS, and OCB2 authenticated encryption can ensure the IND-qCPA security [12] if the internal block ciphers are secure against poly-time quantum attacks. Since no poly-time quantum attack on Pholkos has been proposed and it corresponds to the "quantum secure PRF" defined in [1,31], it can provide IND-qCPA security for such modes. However, the decryption of these modes requires the decryption function of the internal block cipher. Since Pholkos decryption takes about twice as much time as the encryption, its use in these modes may cause non-negligible latency in the decryption process. Therefore, proposing a secure block cipher that is efficient in both the encryption and the decryption is required for using modes of operation such as CBC and XTS on quantum computers in a secure and efficient manner.

1.3 Our Contribution

In this study, we design large-state block ciphers called Ghidle for post-quantum security by inheriting the design principle of Pholkos. Specifically, in this study, for designing Ghidle, we explore how to construct AES-based block ciphers that can provide security and high encryption performance without sacrificing the decryption performance.

The construction of Ghidle is based on the three consecutive AES rounds and an involutive binary matrix instead of the two consecutive AES rounds and a word-wise shuffle which are employed in Pholkos and other AES-based primitives. The constructions of Ghidle ensures that the number of instructions required for encryption and decryption in x86 is the same, which contributes to equalizing the performance in both directions. In the security evaluation, we show that Ghidle is sufficiently secure against several classical attacks. In addition, Ghidle has a key size of 256 bits, thereby making it 128-bit secure against key search using Grover's algorithm. Moreover, the large state size of 256 or 512 bits also provides a "bigger birthday bound" security.

Finally, the performance evaluation shows the high efficiency of Ghidle. We experiment with CPUs of Intel's 10–12 generation in our performance evaluation. As a result, the decryption of the 512-bit block variant Ghidle-512 outperforms Pholkos-512 by about 54%, while the encryption performance remains mostly the same as Pholkos-512. (The 256-bit block variant Ghidle-256 out performs Pholkosby 33% in decryption performance, but performance, whereas the encryption performance is worse than Pholkos-256, a 256-bit block variant of Pholkos.) Furthermore, the AES round functions that compose Ghidle are accounted for the exclusive one for the last round compared to Pholkos. This accelerates the performance of Ghidle even in arm64 where two instructions are required to represent the normal AES round function. Our performance evaluation on mobile devices with arm64-based processors shows that Ghidle-256 outperforms Pholkos-256 by about 32% for decryption, while the encryption performance remains mostly the same as Pholkos-256. And remarkably, Ghidle-512 outperforms Pholkos-512 by about 21% and 53% for both the encryption and the decryption, respectively.

2 Preliminaries

2.1 AES-NI and SIMD Instructions

Most modern processors support instruction set for SIMD (Single Instruction/Multiple Data) which can perform operations vector-wise using data stored in dedicated registers. This allows arithmetic/bitwise operations in parallel and complicated operations like data shuffling to be performed with a single instruction. One of the most famous complex operations that can be efficiently implemented with SIMD instructions is the processing of the most dominant block cipher AES. An instruction set for efficient AES implementation, AES-NI (AES New Instructions set) is supported on most modern x86 processors. AES-NI includes AESENC to perform the round function of the encryption, AESENCLAST for the final round, AESDEC to perform the round function of the decryption, AESIMC to perform the inverse MixColumns, and instructions to support the round key generation. Let ShiftRows, SubBytes, MixColumns, AddRoundKey be SR, SB, MC, AKthe operation of four instructions, AESENC, AESDEC, AESENCLAST and AESDECLAST are expressed as following:

$$\mathsf{AESENC} = \mathrm{AK} \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{SR} \tag{1}$$

$$\mathsf{AESENCLAST} = \mathrm{AK} \circ \mathrm{SB} \circ \mathrm{SR} \tag{2}$$

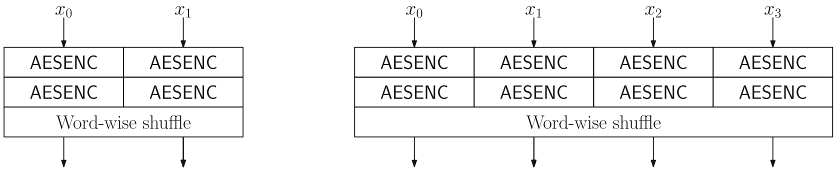
$$\mathsf{AESDEC} = \mathrm{AK} \circ \mathrm{MC}^{-1} \circ \mathrm{SB}^{-1} \circ \mathrm{SR}^{-1} \tag{3}$$

$$\mathsf{AESDECLAST} = \mathrm{AK} \circ \mathrm{SB}^{-1} \circ \mathrm{SR}^{-1} \tag{4}$$

The performance of these instructions in modern processors has evolved. Specifically, In Intel 10th generation and later, the latency of AESENC which was previously 4 is now 3. In addition, since an extra execution port for micro-operations generated from AESENC has added, the throughput which was previously 1 is now 0.5. As a result 6 AESENC can be executed at latency 5 by pipelining.

Processors of arm64 also support AES instructions in NEON instruction set, although its specification differs from instructions in AES-NI. There are four types of AES instructions belonging to NEON: AESE, AESMC, AESD, and AESIMC. AESE and AESD are basically identical to AESENCLAST and AESDECLAST in AES-NI, respectively, except that AK is executed in the beginning instead of the end. AESMC and AESIMC are instructions that execute MC and MC^{-1}, respectively.

In the rest of this paper, the AES instructions introduced in this section refer to the instructions themselves and the functions that perform the operations realized by those instructions.



(a) Basic construction of 256-bit variants

(b) Basic construction of 512-bit variants

Fig. 1. Basic constructions of AES-NI-based primitives with SPN structure

2.2 Large-State Cryptographic Primitives Based on AES-NI

This subsection briefly describes several AES-based primitives that are optimized for implementation by AES instructions.

SPN Primitives. As AES-based primitives with SPN structure, Pholkos [14], Haraka v2 [29] and AESQ [11] are proposed. Pholkos [14] is a family of efficient tweakable block ciphers. Pholkos has a key size of 256 bits, a state size of 256/512/1024 bits, and it is designed to have resistance against Grover's exhaustive search and beyond birthday security. Thus, Pholkos claims 128-bit security even in the quantum setting. The round function is an instantiation of Fig. 1 which employs the word-wise shuffle that can be realized by PBLENDD instructions on x86. Haraka v2 and AESQ also employ the basic construction shown in Fig. 1, albeit the word-shuffle is different for each of them.

Simpira v2. Simpira v2 [23] is a family of Feistel permutations that is proposed for general-purpose use. Simpira v2 however claims only 128-bit security, even for variants with state sizes of 256-bits or larger. Besides, in [30] Kuwakado and Morii have shown that the Even-Mansour construction, underlying construction of Simpira v2 block cipher is vulnerable to a quantum attack using Simon's algorithm.

3 Design Rationale

In this section, we explain our target constructions and their requirements. In the later part of this paper, the part of the AES-based primitives with SPN structure that executes AES instructions is denoted as the AES layer, and the part that executes instructions other than AES instructions is denoted as the linear layer.

Our target constructions are AES-based permutations with SPN structure since it generally requires fewer rounds than Feistel structure to provide sufficient security. To find constructions where both the encryption and the decryption are efficient and there is no gap between their performance, we employ the following two techniques.

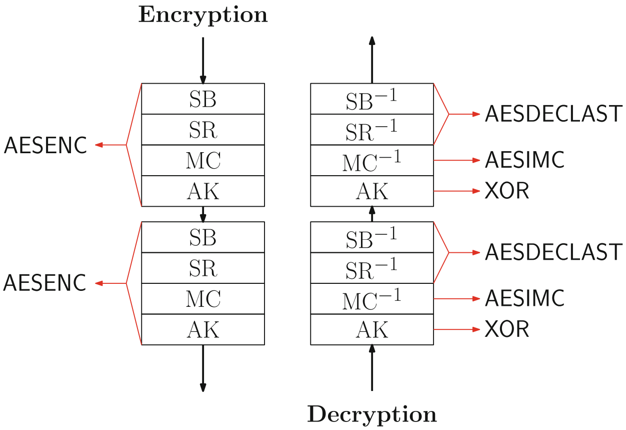


Fig. 2. The difference of the number of instructions between the encryption and decryption

3.1 The AES Layer: Three Consecutive AES Rounds

Problems of Two Consecutive AES Instructions for Decryption. In the case where the AES layer of the encryption is implemented on x86 with two consecutive AESENC, one AESDECLAST, one AESIMC and one XOR instruction are needed to perform the inverse of operations by a single AESENC. Namely, the implementation of the AES layer which consists of two consecutive AES requires the execution of a total of six instructions. Figure 2 indicates why the difference in the number of instructions arises between the encryption and the decryption. Although this number can be reduced to five by incorporating the second MC^{-1} into the key scheduling function, the number of instructions required for the decryption is still twice the number of instructions required for the encryption.

Therefore, the performance of decryption which requires more instructions is lower than the encryption.

Three Consecutive AES Instructions. To keep the performance symmetry between the encryption and decryption, we propose three consecutive AES instructions consisting of not only AESENC but also AESENCLAST as the AES layer, while existing schemes such as Pholkos, Haraka v2, AESQ and Simpira v2 utilize only AESENC instruction. In particular, we clarify the following four requirements for the AES layer to minimize the overhead of the decryption.

Requirement 1. *AESENC should not be executed twice in consecutive order.*

As we mentioned above, the decryption of two consecutive AESENC requires AESIMC and XOR instructions in addition to AESDEC. This makes the number of instructions required to implement the decryption more than the encryption.

Requirement 2. *The third AES instruction executed in the AES layer is limited to AESENCLAST and must not be added to the secret key.*

The inverse of an AESENC requires an AESDECLAST and a XOR. Also, applying the secret key to the third AES instruction in the AES layer requires extra XOR before the first AES instruction in the decryption. To avoid using additional XOR, we have to limit AESENCLAST in the third AES instruction in the AES layer.

Requirement 3. *The round key is not given as the second argument to AESENC.*

Adding the round key to AESENC requires AESIMC. Therefore, the round key is not added in AESENC but only in AESENCLAST.

Requirement 4. *At least one AESENC must be used.*

AESENC is required to increase the diffusion within each 128-bit state. Without this operation, full diffusion is never achieved. Therefore, at least one AESENCis essential.

Optimal AES Layers. There are only two combinations of three consecutive AES rounds satisfying all requirements described in Sect. 3.1. Let AESENC, AESENCLAST, AESDEC and AESDECLAST be two-variable functions which take 128-bit data x as the first argument and round key k as the second argument. Using Eqs. (1)–(4), the functions which represent the two combinations can be expressed as the following two functions: F_γ and F_δ can also be expressed as

$$\begin{aligned} F_\gamma(x,k) &= \mathsf{AESENCLAST}(\mathsf{AESENCLAST}(\mathsf{AESENC}(x,0),k),0),\\ F_\delta(x,k) &= \mathsf{AESENCLAST}(\mathsf{AESENC}(\mathsf{AESENCLAST}(x,k),0),0). \end{aligned}$$

And their inverse are

$$\begin{aligned} F_\gamma^{-1}(x,k) &= \mathsf{AESDECLAST}(\mathsf{AESDEC}(\mathsf{AESDECLAST}(x,k),0),0).\\ F_\delta^{-1}(x,k) &= \mathsf{AESDECLAST}(\mathsf{AESDECLAST}(\mathsf{AESDEC}(x,0),k),0), \end{aligned}$$

We employ these two functions: F_γ and F_δ in the round function of new family of block ciphers Ghidle that we propose in the next section to increase the security against Yoyo-type attacks (details are discussed in Sect. 5.6).

3.2 Linear Layer: SIMD-Friendly Involutional Matrix

Problems of Shuffle Instruction. Shuffle instructions in Haraka v2, Pholkos and AESQ can perform complex data transfers with a small number of instructions. For instance, in x86, Haraka v2 , Pholkos and AESQ use PUNPCK{L,H}DQ, PBLENDD and PSHUFD, respectively. The shuffle layer can be implemented efficiently in environments where these SIMD instructions are available. In particular, PUNPCK{L,H}DQ and PBLENDD are less flexible in terms of generality than PSHUFD, which allows users to set the index of the target permutation, but they can extract specific elements from two xmm registers and store them in a single xmm register with a latency of only one cycle. This can provide the implementation of fast shuffling operations, but these instructions also have some drawbacks. Especially, for PUNPCK{L,H}DQ, the instructions for the inverse operation do not exist in x86. Therefore, this instruction causes an asymmetry in the number of instructions required for the shuffle layer between the encryption and the decryption, and it is not suitable for the cipher we intended to design in this study.

For instance, implementing the forward shuffle for Haraka v2, which is implemented by PUNPCK{L,H}DQ requires 2 and 8 instructions for Haraka-256 and Haraka-512, respectively. On the contrary, implementing the reverse shuffle requires 4 PUNPCK{L,H}DQ for Haraka-256 and 8 PUNPCK{L,H}DQ instructions and 6 PSHUFD for Haraka-512. Note that Haraka v2 was proposed as a hash function, so it is not discussed by the designers.

On the other hand, PBLENDD of Pholkos does not cause such asymmetry of the number of instructions. However, the shuffle by PBLENDD causes implementation problems on arm64, the shuffle of Pholkos-256 can be implemented with only two TRN instructions, whereas the shuffle of Pholkos-512 shuffle requires 10 instructions. Furthermore, the 10 instructions consists of 6 TRN and 4 EXT which increases the number of cycles for throughput by using two distinct instructions. Moreover, even in x86 processors, the shuffle of Pholkos-512 requires 8 instructions. This relatively large number of instructions leaves room for improvement.

Involutory Binary Matrix. Instead of the shuffle operations, we employ an involutory binary matrix multiplication for the linear layer. As an involutory binary matrix, we utilize the one used in a family of lightweight block cipher Midori [2]. Let $(x_0, x_1, x_2, x_3)^T$ be the input and $(y_0, y_1, y_2, y_3)^T$ be the output (each element is a 128-bit subblock), the matrix multiplication can be expressed as:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0\,1\,1\,1 \\ 1\,0\,1\,1 \\ 1\,1\,0\,1 \\ 1\,1\,1\,0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}. \tag{5}$$

The branch number of this matrix is 4. Therefore, at least four of the input and output subblocks are differentially active. In 512-bit block ciphers, this operation can be implemented by 6 XOR instructions while we regard 1 word as a 128-bit data chunk whereas the shuffle operation used in Haraka v2 and Pholkos requires 8 instructions. In addition, this matrix has an involution property, its inverse operation is the same as the forward.

Even if we apply this matrix to 256-bit block ciphers by regarding 1 word as a 64-bit data chunk, this matrix can be implemented 4 PUNPCK{L,H}DQ and 3 XOR instructions. Indeed, for 256-bit block ciphers, shuffle of Haraka v2 and Pholkos requires fewer instructions than this binary matrix. However, the binary matrix can offer higher security than the simple word-wise shuffle operations. Even in the NEON implementation, this matrix for 512-bit block ciphers can

Table 1. Required instructions for shuffle of 256-bit block on x86

	Haraka v2 shuf.	Pholkos shuf.	Matrix (Eq. (5))
Enc.	2 PUNPCK{L,H}DQ	2 PBLENDD	4 PUNPCK{L,H}DQ+ 3 XOR
Dec	4 PUNPCK{L,H}DQ		

Table 2. Required instructions for shuffle of 512-bit block on x86

	Haraka v2 shuf.	Pholkos shuf.	Matrix (Eq. (5))
Enc.	8 PUNPCK{L,H}DQ	8 PBLENDD	6 XOR
Dec	8 PUNPCK{L,H}DQ+ 6 PSHUFD		

be implemented using only 6 XOR instructions. For 256-bit block ciphers, this matrix can be implemented with 2 EXT instructions and 3 XOR instructions. Table 1 and Table 2 summarize the required instructions for shuffle and binary matrix of 256-bit block ciphers and 512-bit block cipher, respectively. We also give a summary of the required instructions on arm64 in Table 3.

4 Ghidle: New AES-NI-Based Block Ciphers

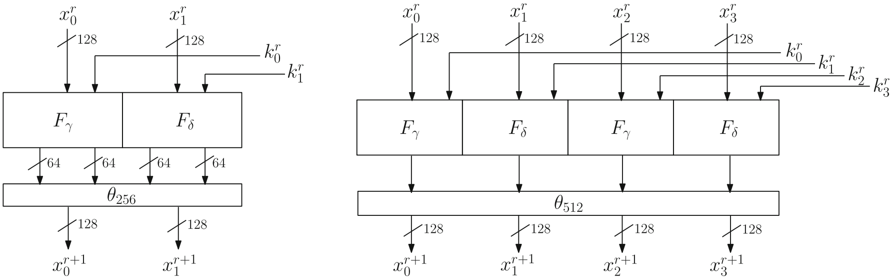
4.1 Specification

Ghidle is a block cipher family consisting of Ghidle-256 with the block size of 256 bits and Ghidle-512 with the block size of 512 bits. Both variants use a 256-bit secret key. The round function and the key scheduling function are shown in Fig. 3 and pseudo codes are shown in Algorithm 1. The round function iterates between $1 \leq r \leq R$, while The key scheduling function iterates between $0 \leq r \leq R$ for generating keys for whitening in addition to round keys. Let the round key of the r-th round be k^r. Note that the value of k^r is $k^r = k_0^r||k_1^r$ for Ghidle-256 and $k^r = k_0^r||k_1^r||k_2^r|||k_3^r$ for Ghidle-512. For the whitening, k^0 is XORed to the plaintext x^0 to generate the initial state x^1, and k^{R+1} is XORed to the state after the final round x^{R+1}. We set the number of rounds R for Ghidle-256 and Ghidle-512 to 7 and 8, respectively, based on the security analysis which is described in detail in the next section.

The round function consists of the AES-based function F_γ and F_δ and a binary matrix multiplication θ_{256} or θ_{512}. The input is divided into two 128-bit substates for Ghidle-256 and four 128-bit substates for Ghidle-512. For each variant, half of the total substates are input to F_γ and the rest to F_δ in their AES layer. This construction of using two distinct functions contributes to the improvement of the security against yoyo attacks. (The details are given in Sect. 6.) The output of the AES layer is multiplied by the matrix of (5) to perform mixing between the substates. In this process, the matrix multiplication is performed over $GF(2^{64})$ and $GF(2^{128})$ for Ghidle-256 and Ghidle-512, respectively. Thus, θ_{256} takes two 128-bit substates as the input and divides

Table 3. Required instructions for shuffle on arm64

	Haraka v2 shuf.	Pholkos shuf.	Matrix (Eq. (5))
256-bit, Enc./Dec.	2 ZIP	2 TRN	2 EXT + 3 XOR
512-bit, Enc./Dec.	8 ZIP	6 TRN + 4 EXT	6 XOR



(a) Ghidle-256

(b) Ghidle-512

Fig. 3. The round function of Ghidle

each 128-bit value into two 64-bit values for the matrix multiplication. While θ_{512} takes four substates as the input vector and directly used it for the matrix multiplication. Besides, the round key is applied at the second execution and the first execution of the round function AESENCLAST in F_γ and F_δ, respectively.

The round keys are generated by the key scheduling function shown in Fig. 4. Let $k_0^0||k_1^0$ be the master key. Then, it is used as the input for Ghidle-256 key scheduling. Although the master key is 256 bits, the input to Ghidle-512 key scheduling is 512 bits. By expanding the master key, we set the input of the Ghidle-512 key scheduling to $k_0^0||k_1^0||k_2^0||k_3^0$ where $k_2^0 = k_0^0$ and $k_3^0 = k_1^0$. π_{unpack} in the key scheduling function of Ghidle-256 is the same as the shuffle of Haraka-256. Let the input of π_{unpack} be $z_0||\ldots||z_7$ where each z_i $(i = 0\ldots7)$ is 32-bit word, π_{unpack} can be expressed as:

$$\pi_{\mathsf{unpack}} : z_0||\ldots||z_7 \mapsto z_0||z_4||z_1||z_5||z_2||z_6||z_3||z_7$$

The key scheduling function uses round constants (c_{2r}, c_{2r+1}) for Ghidle-256 and $(c_{4r}, c_{4r+1}, c_{4r+2}, c_{4r+3})$ for Ghidle-512. These constants are derived from the fraction part of π.

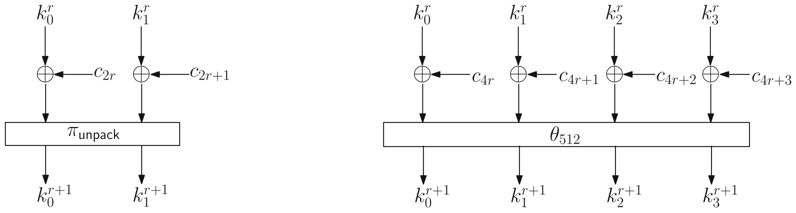
$$c_j = \mathrm{LSB}_{128}((\pi - 3) << 128(j+1)), \forall j = 0, \ldots, 35$$

where j is the LSB_{128} is a function that extracts the least significant 128 bits of the integer portion of the argument.

The decryption can be computed by simply changing F_γ and F_δ to their inverse functions, and reversing the order of the round keys to be added.

Algorithm 1. Ghidle Encryption & Key Scheduling

```
procedure GHIDLE256ENCRYPT(PlainText, SecretKey)
    $x_0^0 || x_1^0 \leftarrow$ PlainText
    $\{k^0, \cdots, k^{R+1}\} \leftarrow$ KEYGEN256(SecretKey)
    $k_0^0 || k_1^0 \leftarrow k^0$
    $\{x_0^1, x_1^1\} \leftarrow \{x_0^0 \oplus k_0^0, x_1^0 \oplus k_1^0\}$
    for $r = 1, \cdots, R$ do
        $k_0^r || k_1^r \leftarrow k^r$
        if $r < R$ then
            $\{x_0^{r+1}, x_1^{r+1}\} \leftarrow \theta_{256}(F_\gamma(x_s^r, k_0^r), F_\delta(x_s^r, k_1^r))$
        else
            $\{x_0^{r+1}, x_1^{r+1}\} \leftarrow \{F_\gamma(x_s^r, k_0^r), F_\delta(x_s^r, k_1^r)\}$
        end if
    end for
    $k_0^{R+1} || k_1^{R+1} \leftarrow k^{R+1}$
    CipherText $\leftarrow (x_0^{R+1} \oplus k_0^{R+1}) || (x_1 \oplus k_1^{R+1})$
    return CipherText
end procedure
----------------------------------------------------------------
procedure GHIDLE512ENCRYPT(PlainText, SecretKey)
    $x_0^0 || x_1^0 || x_2^0 || x_3^0 \leftarrow$ PlainText
    $\{k^0, \cdots, k^{R+1}\} \leftarrow$ KEYGEN512(SecretKey)
    $k_0^0 || k_1^0 || k_2^0 || k_3^0 \leftarrow k^0$
    $\{x_0^1, x_1^1, x_2^1, x_3^1\} \leftarrow \{x_0^0 \oplus k_0^0, x_1^0 \oplus k_1^0, x_2^0 \oplus k_2^0, x_3^0 \oplus k_3^0\}$
    for $r = 1, \cdots, R$ do
        $k_0^r || k_1^r || k_2^r || k_3^r \leftarrow k^r$
        if $r < R$ then
            $\{x_0^{r+1}, x_1^{r+1}, x_2^{r+1}, x_3^{r+1}\} \leftarrow \theta_{512}(F_\gamma(x_0^r, k_0^r), F_\delta(x_1^r, k_1^r), F_\gamma(x_2^r, k_2^r), F_\delta(x_3^r, k_3^r))$
        else
            $\{x_0^{r+1}, x_1^{r+1}, x_2^{r+1}, x_3^{r+1}\} \leftarrow \{F_\gamma(x_0^r, k_0^r), F_\delta(x_1^r, k_1^r), F_\gamma(x_2^r, k_2^r), F_\delta(x_3^r, k_3^r)\}$
        end if
    end for
    $k_0^{R+1} || k_1^{R+1} || k_2^{R+1} || k_3^{R+1} \leftarrow k^{R+1}$
    CipherText $\leftarrow (x_0^{R+1} \oplus k_0^{R+1}) || (x_1^{R+1} \oplus k_1^{R+1}) || (x_2^{R+1} \oplus k_2^{R+1}) || (x_3^{R+1} \oplus k_3^{R+1})$
    return CipherText
end procedure
----------------------------------------------------------------
procedure KEYGEN256(SecretKey)
    $k^0 \leftarrow$ SecretKey
    for $r = 0, \cdots, R$ do
        $k^{r+1} \leftarrow \pi_{\mathsf{unpack}}(k^r)$
    end for
    return $\{k^0, \cdots, k^{R+1}\}$
end procedure
----------------------------------------------------------------
procedure KEYGEN512(SecretKey)
    $k^0 \leftarrow$ SecretKey
    for $r = 0, \cdots, R$ do
        $k_0^r || k_1^r || k_2^r || k_3^r \leftarrow k^r$
        $\{k_0^{r+1}, k_1^{r+1}, k_2^{r+1}, k_3^{r+1}\} \leftarrow \theta_{512}(k_0^r, k_1^r, k_2^r, k_3^r)$
        $k^{r+1} \leftarrow k_0^{r+1} || k_1^{r+1} || k_2^{r+1} || k_3^{r+1}$
    end for
    return $\{k^0, \cdots, k^{R+1}\}$
end procedure
```



(a) Ghidle-256 (b) Ghidle-512

Fig. 4. The key scheduling function of Ghidle

4.2 Claimed Security

For both variant of Ghidle, we claim 256-bit and 128-bit security in classical and quantum setting, respectively, where the available data complexity of Ghidle-256 and Ghidle-512 for each key is limited to 2^{128} and 2^{256}, which comes from the bigger birthday bounds of each variants for mode of operation.

5 Security Evaluation

In this section, we evaluate the security against differential/linear, truncate differential, impossible-differential, integral attacks, boomerang, Rectangle, Yoyo, mixuture differential and DS-Meet-in-the-middle attacks for Ghidle as well as quantum attacks.

5.1 Differential and Linear Cryptanalysis

Differential/linear attacks are the most basic attacks on block ciphers. In this paper, we employed the MILP (Mixed-Integer Linear Programming)-based method proposed in [32] for the evaluation. As a result, the lower bound of active S-boxes of Ghidle-256 and Ghidle-512 reach 27 in 2 rounds and 51 in 3 rounds, respectively. Since the differential/linear probability of AES S-box is 2^{-6} and available data complexity of Ghidle-256 and Ghidle-512 for each key is limited to 2^{128} and 2^{256}, Ghidle-256 and Ghidle-512 are secure against differential/linear attacks after 2 and 3 rounds, respectively.

5.2 Truncated Differential Cryptanalysis

The notion of truncated differential cryptanalysis was introduced by Knudsen [27] in which instead of concrete values, patterns of differences (zero or non-zero) are considered. In our analysis, 3-round Ghidle-256 a truncated differential trail can be constructed with probability $2^{-96} \times (2^{-16})^4 = 2^{-160}$. For Ghidle-512, we can extend the deterministic truncated differential of 3-round by prepending one round at the beginning. As a result, we can construct the 4-round trail for Ghidle-512 with probability $2^{-96} \times 2^{-32} \times 2^{-32} \times (2^{-16})^4 = 2^{-224}$.

5.3 Impossible-Differential Cryptanalysis

Impossible-differential cryptanalysis [9,26] exploits the differences holding with probability 0 (impossible differences). For both Ghidle-256 and Ghidle-512, our MILP-aided search verified that there is no byte-wise impossible differential in the case where only one byte is active in each of the input and output. As a result, we find 4 and 5-round impossible differentials on Ghidle-256 and Ghidle-512, respectively. However, due to the limitation of available data, these distinguishers can not be directly used for attacks on Ghidle-256 and Ghidle-512, because it requires around 2^{256-8} and 2^{512-8} pairs in which there is no difference except one byte to mount distinguishing attacks. We believe that Ghidle-256 and Ghidle-512 are secure against these types of attacks after 4 and 5 rounds at worst under the data limitation, respectively.

5.4 Integral Cryptanalysis

Integral attacks [17,28] use a set of plaintexts, which has a constant value at some bit while the other part takes all possible values. We use the MILP-based method proposed by Xiang [36], and then we search the case where only one input byte is constant, and the remaining bytes are active, to estimate the upper bounds of the number of rounds for integral distinguisher for Ghidle-256 and Ghidle-512. As a result, we find 2 and 3-round integral distinguishers on Ghidle-256 and Ghidle-512, respectively. Besides, as available data complexity is limited to 2^{128} and 2^{256}, Ghidle-256 and Ghidle-512 are secure against integral attacks after 2 and 3 rounds, respectively.

5.5 Boomerang and Rectangle Attack

In boomerang cryptanalysis, a primitive E is considered as $E_1 \circ E_0$ and two independent differential trails through sub-cipher E_0 and E_1 are combined to find boomerang quartet through E [35]. In particular, if there is a differential trail $\alpha \rightarrow \beta$ with probability p through E_0 and a differential trail $\gamma \rightarrow \delta$ with probability q through E_1, then a boomerang quartet can be found with probability p^2q^2. In our analysis, in the secret key-setting, boomerang distinguisher can exist for 2-round and 3-round of Ghidle (both variants) with probability 2^{-72} and 2^{-396} respectively.

Rectangle attack [10,25] convert the setting of boomerang attack from adaptive chosen plaintext/ciphertext to chosen plaintext at the expense of reducing the probability of finding quartets. The notion of the attack is quite similar to the boomerang, in which two shorter trails are combined to find a trail over large number of rounds. If the primitive E is an n-bit block cipher, then using the rectangle attack a quartet can be found with probability $2^{-n-1}p^2q^2$ which ensures that the possibility of finding a quartet is much lower than using the boomerang attack. For 2-round Ghidle-256 and Ghidle-512, a right quartet can be found using this attack with probability 2^{-329} and 2^{-585}, respectively.

5.6 Yoyo Attack

In the Yoyo attack [8], initially a pair of plaintexts is chosen that satisfies a certain property. Words are swapped between the corresponding ciphertexts of these plaintexts and the resulting ciphertexts are decrypted to obtain a new pair of plaintext. It is expected that the new pair should also possess the same property as the initial one. Rønjom *et al.* extended the Yoyo attack for SPN ciphers and devised a deterministic distinguisher for generalized 2-round SPN [33]. Saha *et al.* adapted the result on 2-round SPN to devise a probabilistic distinguisher for 9-round AESQ permutation [34]. Both of these attacks exploit the underlying megasbox (cf. [18]). Note that, as both F_γ and F_δ are used in the round function, the MC operations are not applied at the same position in the substates. This, in turn, allows restricting the span of megasbox to just one round. For 2-round Ghidle-256, a deterministic distinguisher can be devised whereas a probabilistic distinguishing attack can be mounted on 2-round Ghidle-512. The (data, time, memory) complexity of the attacks on 2-round Ghidle-512 is ($2^{27.41}$, $2^{27.41}$, *negligible*). For more details on the attack, refer to [34, Algorithm 3].

In [33], the notion of impossible differential yoyo distinguisher is introduced for 3-round SPN. However, this distinguisher is not a generic one and depends highly on the linear layer. In [3], this strategy is adapted to mount distinguishing attacks on 9-round ForkAES. Similar type of attack can also be mounted on Ghidle-512. If the operations till initial AESENC in each substates are omitted, then 3-round Ghidle-512 can be considered as a 3-round SPN where the megasboxes correspond to the substitution layer. Thus an impossible differential yoyo distinguisher can be devised for 3-round Ghidle-512 (without the initial AESENC) with ($2^{252.4}$, $2^{252.4}$, *negligible*) complexity.

5.7 Mixture Differential Cryptanalysis

In mixture differential [20] it is shown that ciphertext difference produced by a pair of plaintexts and their mixture (new plaintext pair is produced by swapping some words between the original ones) counterpart are related to each other. Initially it was deterministic in nature and was used to devise 4-round deterministic distinguisher on AES. Later, a probabilistic version of this technique was also proposed in 2019 [21]. Some other variants of mixture differential are also proposed which are also probabilistic [4–6].

If AESENCLAST ∘ AESENC of the initial F_γ and AESENC ∘ AESENCLAST of the initial F_δ are omitted, then the deterministic mixture differential attack of [20], can be adapted for 2-round Ghidle-256 and Ghidle-512. However, due to the binary matrix multiplication operation and positional difference of MC in F_γ and F_δ, the probabilistic mixture differential attacks can not be directly adapted to our construction.

5.8 DS-Meet-in-the-Middle Attacks

The DS-MITM technique [19] is a powerful attack on AES. In this attack, the aim is to determine a set of possible sequences of values. Specifically, by traversing one input byte and setting other input bytes as random constants, after 4 rounds of AES, there exists a byte whose values only depend on 25 guessed bytes. In other words, under the input set, there are at most $(2^8)^{25}$ possible sequences of $2^8 = 256$ values for this byte. However, for a random permutation, the number of sequences is $(2^8)^{256}$. Therefore, the time complexity of this distinguisher is $2^{8\times25+8} = 2^{208}$ and the data complexity is only 2^8. We performed similar analysis for Ghidle-256 and could only found a distinguisher for 3 rounds. In addition, we can append 1 round for the key recovery. Hence, the security margin is 3 rounds for Ghidle-256. For Ghidle-512, we also found a DS-MitM distinguisher for up to 4 rounds and could append 1 round for the key recovery. Therefore, the security margin is also 3 rounds for Ghidle-512.

5.9 Quantum Attacks

For block ciphers with a key of size k, a quantum adversary can perform an exhaustive key search using Grover's algorithm in time approximately $2^{k/2}$. For Ghidle, it will require approximately 2^{128} iterations, for both 256 and 512-bit variants.

Considering that the design of Ghidle consists of the operations of AES, we believe the quantum attacks on AES would be best suited for Ghidle. One of the complete quantum security analysis of AES was presented in [13]. The quantum DS-MITM attack on AES in [13] reaches 8-rounds in the case where the key size is double of the state size. For Ghidle-512, in the classical settings there exists a DS-MITM distinguisher for up to 4 rounds and 1 more round could be appended for key recovery. Due to the binary matrix multiplication operation and positional difference of MC in F_γ and F_δ, we believe the quantum security margin will still be almost similar to that of the classical security margin and full round Ghidle-512 should be safe from the quantum DS-MITM attack. We know that the quantum differential and linear attacks enjoys quadratic speed up [24]. From the lower bounds on the number of active S-boxes computed in Table 7, we believe Ghidle-256 and Ghidle-512 are secure against quantum differential attacks after 3 and 4 rounds respectively. These results show that, for now, the full round Ghidle is quantum secure.

6 Performance Evaluation

In this section, we evaluate software performance of Ghidle-256 and Ghidle-512 on x86 and latest arm64-based processors. Our evaluations use the available source code at GitHub[1] to evaluate the cycle counters, *i.e.*, cycles per byte (cpb), in the target primitives.

All our evaluations in x86 were performed on the following environments: the Ice Lake platform has an Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30 GHz; the Tiger Lake platform has an Intel(R) Core(TM) i7-1165G7 CPU @ 2.80 GHz; the

[1] https://github.com/seb-m/cycles.

Table 4. Benchmarks for single block encryption/decryption of 256-bit ciphers (all values are given as cpb), where BC and TBC mean block cipher and tweakable block cipher, respectively

Primitive	Security	Ice Lake		Tiger Lake		Alder Lake	
		Enc	Dec	Enc	Dec	Enc	Dec
Ghidle-256 (BC)	256 bits	**3.79**	**3.85**	**3.79**	**3.84**	**3.83**	**3.85**
Pholkos-256 (TBC)	256 bits	2.40	5.76	2.40	5.74	2.37	5.71

Table 5. Benchmarks for single block encryption/decryption of 512-bit ciphers (all values are given as cpb)

Algorithm	Security	Ice Lake		Tiger Lake		Alder Lake	
		Enc	Dec	Enc	Dec	Enc	Dec
Ghidle-512 (BC)	256 bits	**2.00**	**1.99**	**2.00**	**1.99**	**1.95**	**1.98**
Pholkos-512 (TBC)	256 bits	2.00	4.29	2.00	4.28	2.02	4.30

Alder Lake platform has an Intel(R) Core(TM) i9-12900K CPU @ 3.20 GHz on a performance-core (P-core) and 2.40 GHz on an efficient-core (E-core); with the Turbo Boost technology disabled off for all our evaluations. We note here that the P-core has been specified on the Alder Lake platform.

For the evaluation in arm64, we use three up-to-date mobile devices with arm64-based processors: Google Pixel 7, Apple iPhone 14, and iPad Pro. Although the details of the CPU of the iPhone/iPad Pro are not publicly available, Tensor2, the CPU of the Pixel 7, is reported to be equipped with two Cortex-X1 cores, two Cortex-A76 cores, and four Cortex-A55 cores. All instances of Ghidle and Pholkos are implemented in C with intrinsic functions.

6.1 Performance on X86

We ran benchmarks on Ghidle and Pholkos [14], and compare their performance. Although Haraka v2 has similar construction to Ghidle and Pholkos, we exclude Haraka v2 from our evaluation because it is proposed as a hash function, not a block cipher. In all evaluations on x86, we measure the cycle per byte for single block encryption/decryption by iterating 1.25E8 times the execution of the target algorithm and taking the average cycle per byte.

The results are shown in Table 4 and 5. We find that Ghidle-256 and Ghidle-512 are equally fast in the encryption and decryption performances as we aim for it in this paper. The decryption speed of Ghidle-256 and Ghidle-512 are about 33% and 53% faster than Pholkos-256 and Pholkos-512, respectively. On the encryption speed, Ghidle-512 is almost same as Pholkos-512, while that of Ghidle-256 is slightly slower than Pholkos-256.

Saturnin claims 3.5 cpb performance on x86 in the proposed paper [15]. However, to reach the claimed performance of 3.5 cpb, 8 instances must be executed in parallel by bit-slice manner. Therefore, it can be concluded that Ghidle is more suitable for x86 environments than Saturnin.

6.2 Performance on Arm64

We ran the benchmark also on arm64. In all evaluations on arm64, we measure the average gigabit per second required to process one block when processing a sufficiently large number of blocks in parallel.

The result of the measurements is shown in Table 6. As a result, the decryption of Ghidle-256 outperforms Pholkos-256 by about 32%, while the encryption performance of Ghidle-256 is almost the same as Pholkos-256. On the other hand, both the encryption and decryption of Ghidle-512 outperform Pholkos-512 by about 21% and 53%, respectively.

This result can be attributed to the small number of required instructions for the NEON implementation. The total number of the AES round execution which can not be parallelized in Ghidle-256 and Ghidle-512 are 24 and 21(#rounds × 3), respectively. On the other hand, in Pholkos-256 and Pholkos-512, at least 16 and 20 AES round execution can not be parallelized. Since the AES round in the NEON implementation requires two distinct instructions: AESE and AESMC, the number of required instructions for execution of the AES round increased up to twice. That is to say, the number of instructions for consecutive AES rounds of Pholkos-256 and Pholkos-512 are 32 and 40, respectively, in arm64. On the other hand, 2/3 of the AES rounds in Ghidle are the last round function which can be implemented by only AESE. Therefore, the increase in the number of instructions is moderate in the arm64 transplant. Since 1/3 of the total AES rounds corresponding to the AESENC of x86 in Ghidle and the NEON implementations requires two instructions for its representation, the number of AES rounds that can not be parallelized in the NEON implementation of Ghidle is obtained by $(1 + 1/3) \times (\textit{number of AES rounds that can not be parallelized})$.

As a result, the number of instructions for consecutive AES executions of Ghidle-256 and Ghidle-512 increases only up to 32 and 28, respectively. Thus, the number of AES instructions for Ghidle-512 fewer than that of Pholkos-512, while the number of AES instructions for Pholkos-256 and Ghidle-256 is identical. In addition, the number of instructions for the shuffle layer of Ghidle-512 is fewer than that of Pholkos-512, which further reduced the number of execution cycles of Ghidle-512 compared to Pholkos-512. These facts contribute to the advantage of Ghidle as shown in the results.

Table 6. Benchmarks on the Pixel 7, iPhone 14, and iPad Pro (all values are given as Gbps).

Algorithm	Pixel 7		iPhone 14		iPad Pro	
	Enc	Dec	Enc	Dec	Enc	Dec
Ghidle-256	**18.40**	**12.76**	**32.08**	**24.71**	**34.64**	**26.72**
Pholkos-256	18.24	8.66	34.83	19.80	37.22	21.41
Ghidle-512	**12.54**	**15.15**	**24.53**	**27.58**	**26.62**	**29.76**
Pholkos-512	9.89	6.98	19.98	13.67	21.47	14.75

7 Conclusion

In this paper, we proposed a new family of block ciphers called Ghidle which can ensure post-quantum security. The construction solves the performance gap between the encryption and decryption on AES instructions-enabled environments that is observed in existing AES-NI-based cryptographic primitives with the SPN structure. The result of the performance evaluation shows that Ghidle has the same encryption and decryption speed on x86. The result also shows that Ghidle-512 outperform Pholkos-512 by about 54% in the decryption, while the encryption of Ghidle-512 is mostly the same as Pholkos-512. Besides, our performance evaluation on mobile devices with arm64-based processors shows that Ghidle-256 outperforms Pholkos-256 by about 32% for decryption while the encryption remains as high performance as Pholkos-256. Remarkably, Ghidle-512 outperforms Pholkos-512 by about 21% and 53% for both encryption and decryption, respectively.

Acknowledgments. Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031. This research was in part conducted under a contract of “Research and development on new generation cryptography for secure wireless communication services” among “Research and Development for Expansion of Radio Wave Resources (JPJ000254)”, which was supported by the Ministry of Internal Affairs and Communications, Japan. This result is also obtained from the commissioned research (No.05801) by National Institute of Information and Communications Technology (NICT), Japan.

Appendix

Lower Bounds on the Number of Active S-Boxes

Table 7 shows lower bounds on the number of active S-boxes for each round of Ghidle.

Table 7. Lower bounds of the number of active S-boxes for differential/linear attacks.

Round Number	1	2	3	4	5	6	7
Min. # of AS (Ghidle-256)	6	27	51	87	101	119	145
Min. # of AS (Ghidle-512)	6	27	51	105	147	168	204

Integral Distinguisher

Table 8 shows detailed integral distinguishers of Ghidle. A, B, and C represent active byte, constant, and balanced byte, respectively.

Table 8. Integral distinguisher on Ghidle-256 and Ghidle-512

Cipher	Integral distinguisher
Ghidle-256	In: (CAAAAAAAAAAAAAAA, AAAAAAAAAAAAAAAA)
	Out: (BBBBBBBBBBBBBBBB, BBBBBBBBBBBBBBBB)
Ghidle-512	In: (CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)
	Out: (BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB, BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB)

Impossible Differential Distinguishers

Figure 5 and 6 show impossible differential distinguishers of the 4-round Ghidle-256 and 5-round Ghidle-512, respectively. Table 9 show the input and output of the impossible differentials in byte-wise representation.

Table 9. Impossible differential distinguisher on Ghidle-256 and Ghidle-512

Cipher	Round	Impossible differential
Ghidle-256	4	In:(1000000000000000, 0000000000000000)
		Out:(0000000000000000, 1000000000000000)
Ghidle-512	5	In:(1000000000000000, 0000000000000000, 0000000000000000, 0000000000000000)
		Out:(0000000000000000, 1000000000000000, 0000000000000000, 0000000000000000)

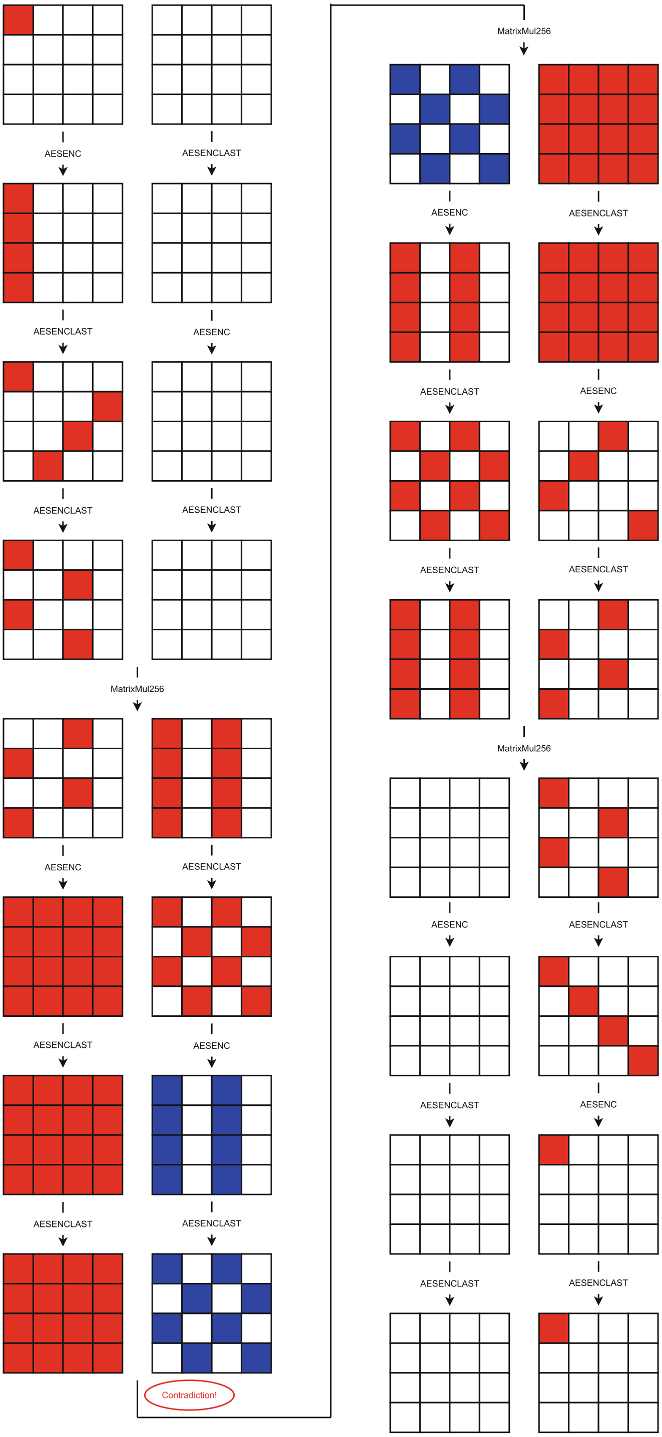


Fig. 5. Impossible differential distinguisher of the 4-round Ghidle-256

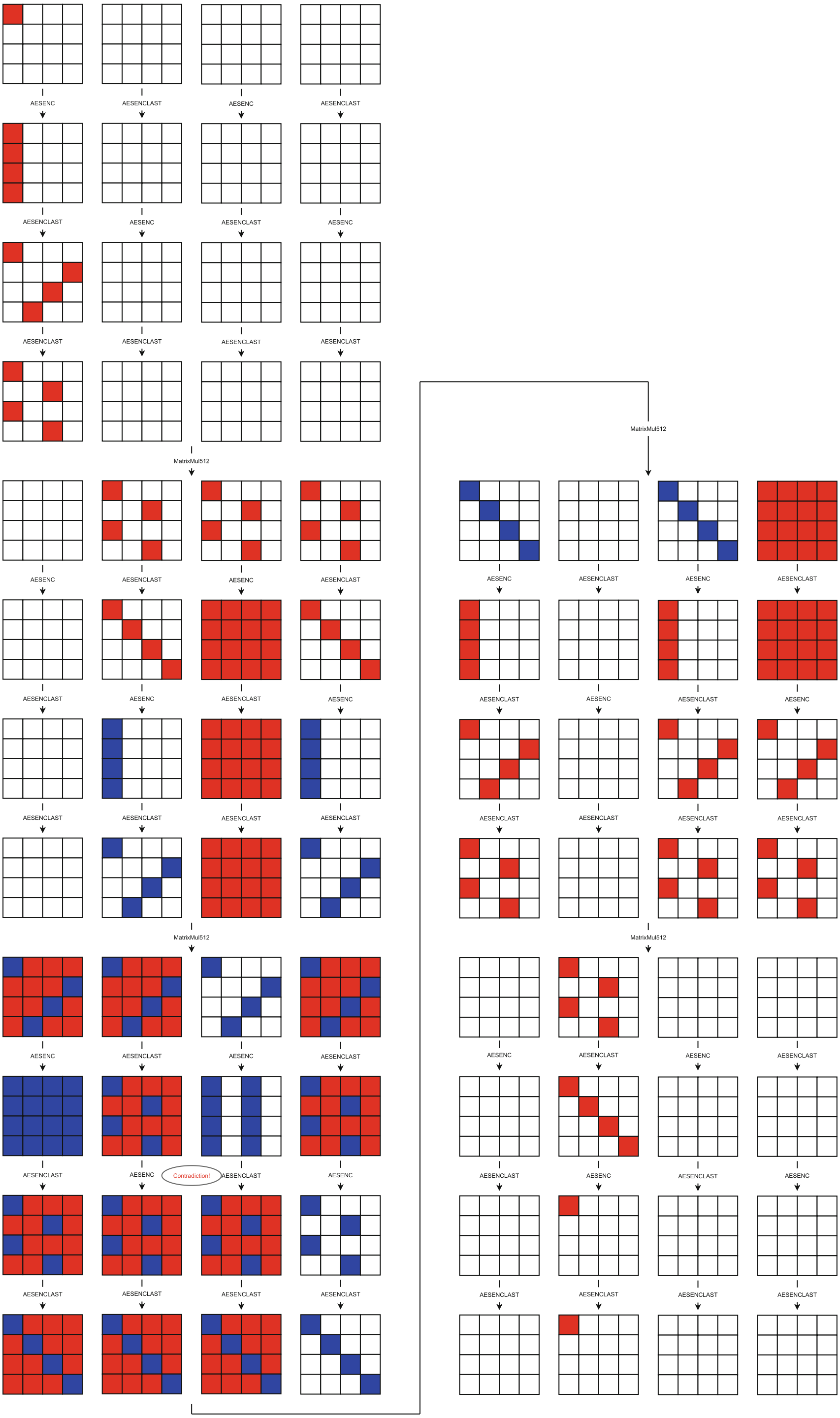
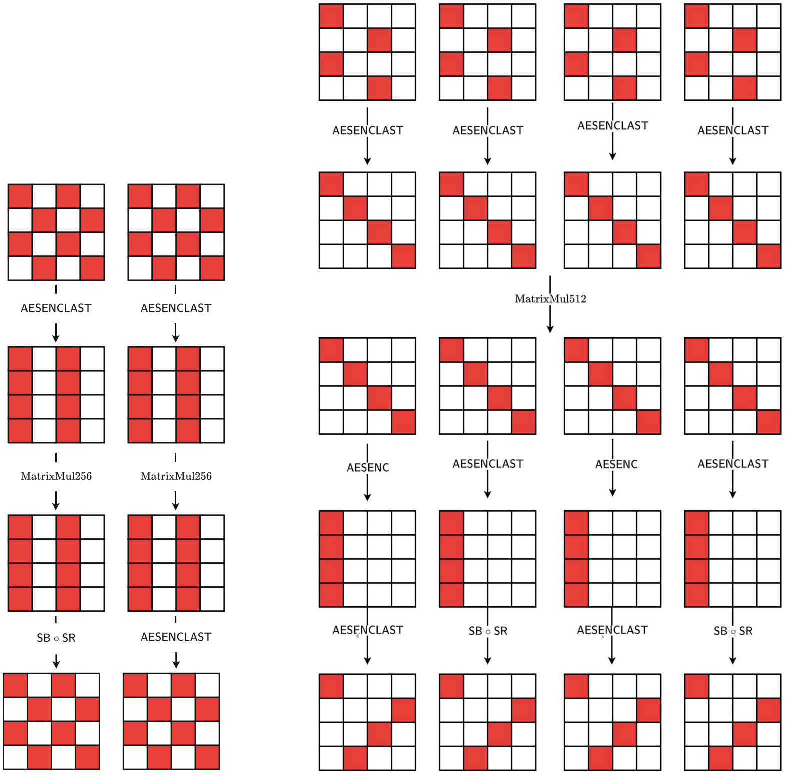


Fig. 6. Impossible differential distinguisher of the 5-round Ghidle-512

Yoyo Attacks

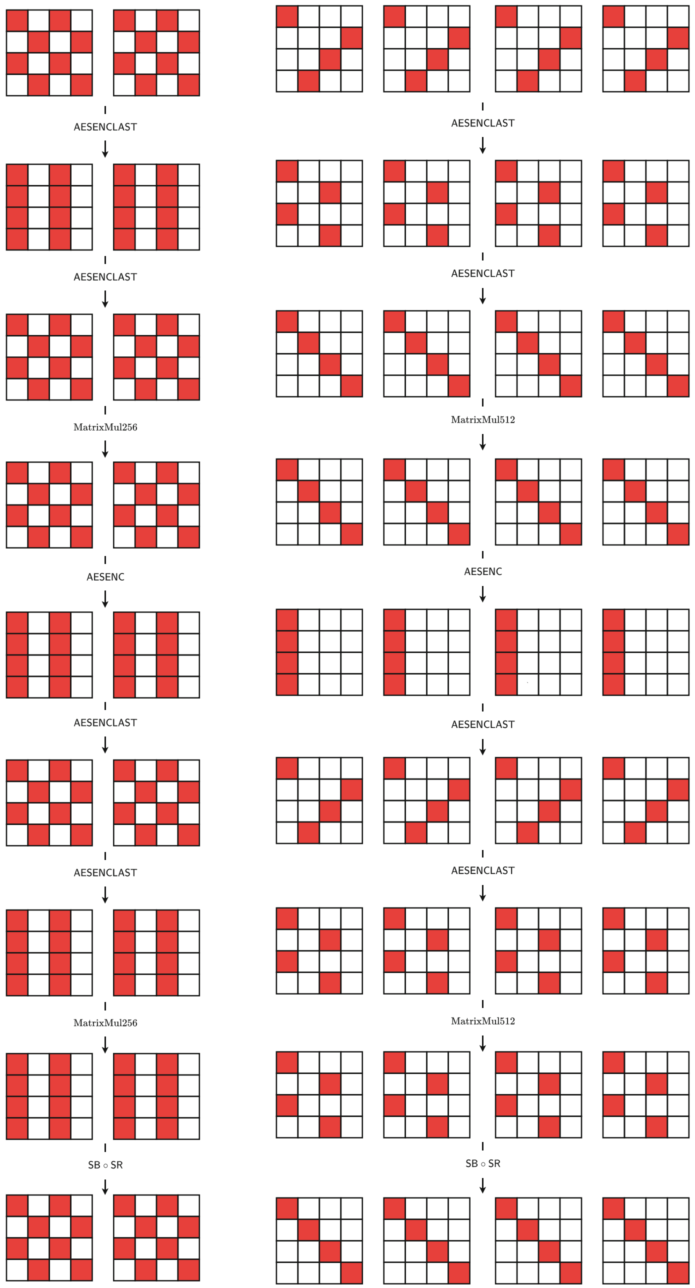


(a) MegaSbox of Ghidle-256

(b) MegaSbox of Ghidle-512

Fig. 7. MegaSbox of Ghidle. The 🟥 bytes corresponds to a single megasbox for each variant of Ghidle. For Ghidle-512, megasbox starting from the first inverse diagonal is shown. There are three more megasboxes corresponding to the three remaining inverse diagonals. In Ghidle-256, there are two parallel megasboxes. (Color figure online)

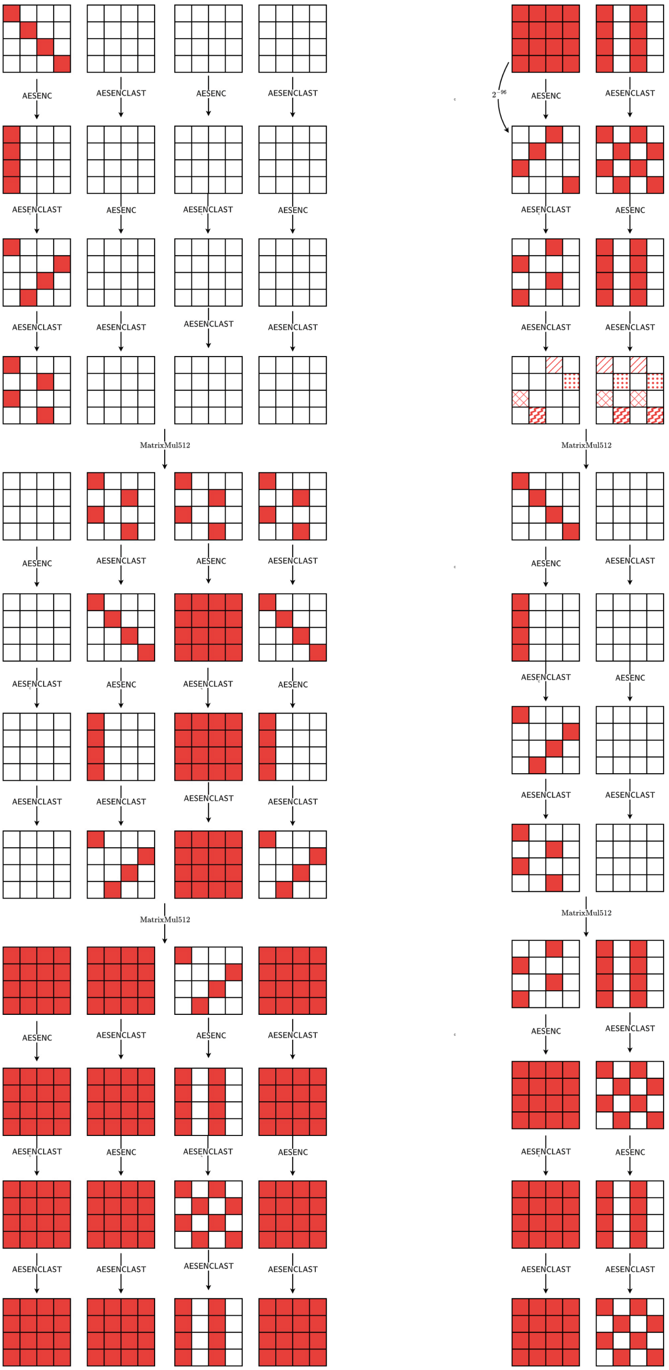
The megasbox of Ghidle is shown in Fig. 7. Here, we show that more number of rounds can be penetrated using yoyo attack if only one of F_γ or F_δ is used in the round function of Ghidle. The underlying megasbox for the case when round function is comprised of only F_γ is shown in Fig. 8. Attacks similar to the ones in [34] can be mounted on 4-round Ghidle-256 and 4-round Ghidle-512 in the secret-key setting by exploiting the underlying megasbox. The (data, time, memory) complexity of the attacks on Ghidle-256 and Ghidle-512 are (2^{29}, 2^{29}, *negligible*) and ($2^{26.41}$, $2^{26.41}$, *negligible*) respectively. Impossible differential yoyo distinguishing attack [33] can also be mounted on Ghidle-512. If the initial AES-ENC is omitted, then 6-round Ghidle-512 can be considered as a 3-round SPN



(a) MegaSbox of Ghidle-256

(b) MegaSbox of Ghidle-512

Fig. 8. MegaSbox of Ghidle when only F_γ is used in the round function. The 🟥 bytes corresponds to a single megasbox for each variant of Ghidle. For Ghidle-512, megasbox starting from the first inverse diagonal is shown. There are three more megasboxes corresponding to the three remaining inverse diagonals. In Ghidle-256, there are two parallel megasboxes. (Color figure online)



(a) Truncated differential trail of 3-round Ghidle-512

(b) Truncated differential trail of 3-round Ghidle-256

Fig. 9. Truncated differential trails of Ghidle

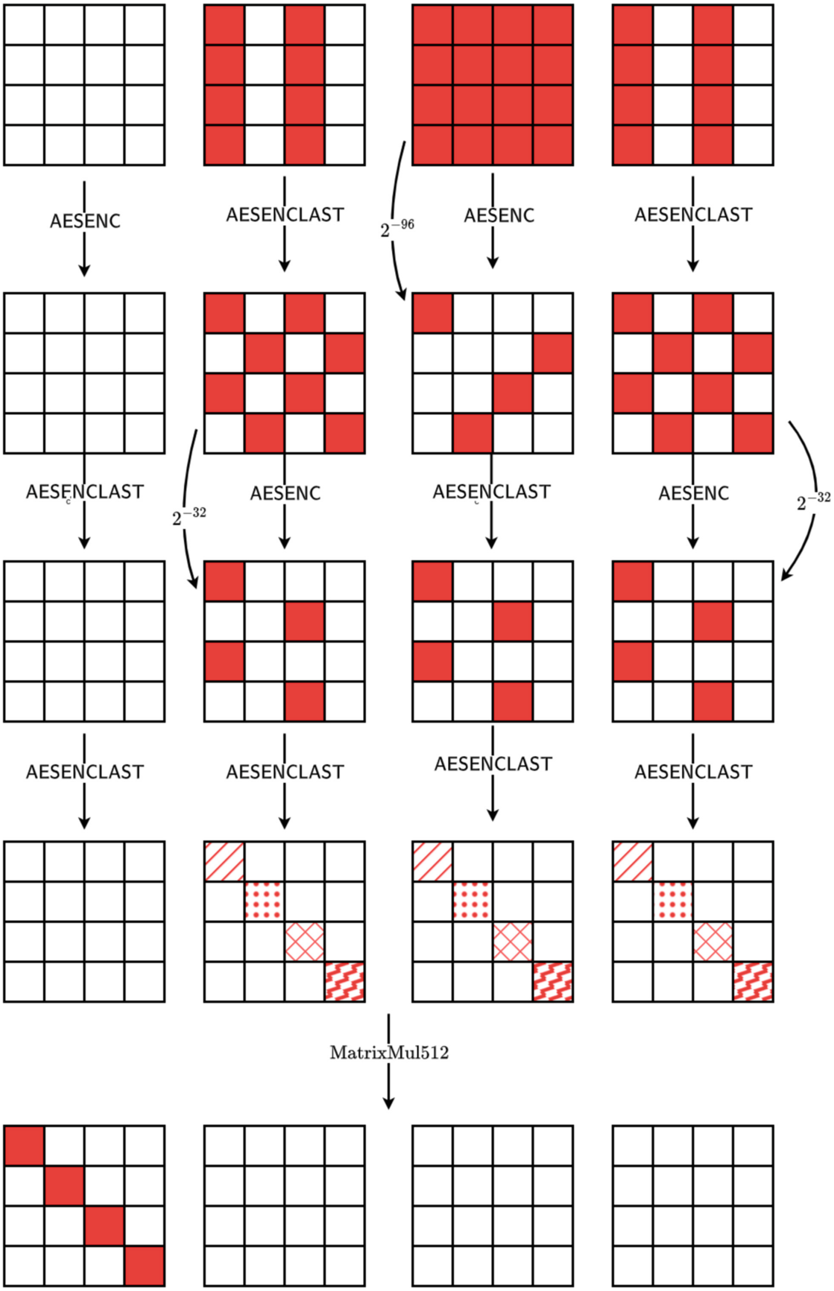


Fig. 10. One round truncated differential trail of Ghidle-512 that can be prepended to 3-round deterministic trail.

where the megasboxes correspond to the substitution layer. Thus a impossible differential yoyo distinguisher can be devised for 6-round Ghidle-512 (without the initial AESENC) with ($2^{252.4}$, $2^{252.4}$, *negligible*) complexity.

Truncated Differential Trails

The deterministic truncated differential trail of 3-round Ghidle-512 is shown in Fig. 9a. The trail can be extended by prepending one round at the beginning as shown in Fig. 10. The bytes with similar pattern after the last AESENCLAST operation are required to hold the same value which can occur with probability $2^8 \times 2^{-8} \times 2^{-8} \times 2^{-8} = 2^{-16}$ for three randomly generated bytes. Hence, the overall probability of the 4-round trail is $2^{-96} \times 2^{-32} \times 2^{-32} \times (2^{-16})^4 = 2^{-224}$. Similarly, for 3-round Ghidle-256 a truncated differential trail can be constructed with probability $2^{-96} \times (2^{-16})^4 = 2^{-160}$.

References

1. Anand, M.V., Targhi, E.E., Tabia, G.N., Unruh, D.: Post-quantum security of the CBC, CFB, OFB, CTR, and XTS modes of operation. In: Takagi, T. (ed.) PQCrypto 2016. LNCS, vol. 9606, pp. 44–63. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29360-8_4
2. Banik, S., et al.: Midori: a block cipher for low energy. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 411–436. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_17
3. Banik, S., et al.: Cryptanalysis of ForkAES. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 43–63. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21568-2_3
4. Bardeh, N.G.: A key-independent distinguisher for 6-round AES in an adaptive setting. Cryptology ePrint Archive, Paper 2019/945 (2019)
5. Bardeh, N.G., Rønjom, S.: The exchange attack: *how to distinguish six rounds of AES with* $2^{88.2}$ *chosen plaintexts*. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 347–370. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_12
6. Bardeh, N.G., Rønjom, S.: Practical attacks on reduced-round AES. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2019. LNCS, vol. 11627, pp. 297–310. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_15
7. Bernstein, D.J.: Some challenges in heavyweight cipher design. In: Dagstuhl Seminar on Symmetric Encryption, Dagstuhl, Germany, vol. 15 (2016)
8. Biham, E., Biryukov, A., Dunkelman, O., Richardson, E., Shamir, A.: Initial observations on skipjack: cryptanalysis of skipjack-3XOR. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 362–375. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48892-8_27
9. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. J. Cryptol. **18**(4), 291–311 (2005)
10. Biham, E., Dunkelman, O., Keller, N.: The rectangle attack — rectangling the serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_21

11. Biryukov, A., Khovratovich, D.: PAEQ: parallelizable permutation-based authenticated encryption. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) ISC 2014. LNCS, vol. 8783, pp. 72–89. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13257-0_5
12. Boneh, D., Zhandry, M.: Secure signatures and chosen ciphertext security in a quantum computing world. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 361–379. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_21
13. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Trans. Symmetric Cryptol. **2019**(2), 55–93 (2019)
14. Bossert, J., List, E., Lucks, S., Schmitz, S.: Pholkos – efficient large-state tweakable block ciphers from the AES round function. In: Galbraith, S.D. (ed.) CT-RSA 2022. LNCS, vol. 13161, pp. 511–536. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-95312-6_21
15. Canteaut, A., et al.: Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. IACR Trans. Symmetric Cryptol. **2020**(S1), 160–207 (2020)
16. Chen, L., et al.: Report on post-quantum cryptography, vol. 12. US Department of Commerce, National Institute of Standards and Technology (2016)
17. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052343
18. Daemen, J., Lamberger, M., Pramstaller, N., Rijmen, V., Vercauteren, F.: Computational aspects of the expected differential probability of 4-round AES and AES-like ciphers. Computing **85**(1–2), 85–104 (2009)
19. Demirci, H., Selçuk, A.A.: A meet-in-the-middle attack on 8-round AES. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 116–126. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_7
20. Grassi, L.: Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. IACR Trans. Symmetric Cryptol. **2018**(2), 133–160 (2018)
21. Grassi, L.: Probabilistic mixture differential cryptanalysis on round-reduced AES. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 53–84. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38471-5_3
22. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, 22–24 May 1996, pp. 212–219. ACM (1996)
23. Gueron, S., Mouha, N.: Simpira v2: a family of efficient permutations using the AES round function. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 95–125. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_4
24. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. IACR Trans. Symmetric Cryptol. **2016**(1), 71–94 (2016)
25. Kelsey, J., Kohno, T., Schneier, B.: Amplified boomerang attacks against reduced-round MARS and serpent. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 75–93. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44706-7_6
26. Knudsen, L.: Deal - a 128-bit block cipher. In: NIST AES Proposal (1998)
27. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60590-8_16

28. Knudsen, L., Wagner, D.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45661-9_9
29. Kölbl, S., Lauridsen, M.M., Mendel, F., Rechberger, C.: Haraka v2 - efficient short-input hashing for post-quantum applications. IACR Trans. Symmetric Cryptol. **2016**(2), 1–29 (2016)
30. Kuwakado, H., Morii, M.: Security on the quantum-type even-mansour cipher. In: ISITA, pp. 312–316. IEEE (2012)
31. Maram, V., Masny, D., Patranabis, S., Raghuraman, S.: On the quantum security of OCB. IACR Trans. Symmetric Cryptol. **2022**(2), 379–414 (2022)
32. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C.-K., Yung, M., Lin, D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34704-7_5
33. Rønjom, S., Bardeh, N.G., Helleseth, T.: Yoyo tricks with AES. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 217–243. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_8
34. Saha, D., Rahman, M., Paul, G.: New Yoyo tricks with AES-based permutations. IACR Trans. Symmetric Cryptol. **2018**(4), 102–127 (2018)
35. Wagner, D.: The boomerang attack. In: Knudsen, L. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48519-8_12
36. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24

Quantum Algorithm for Finding Impossible Differentials and Zero-Correlation Linear Hulls of Symmetric Ciphers

Huiqin Chen[1,4], Yongqiang Li[1,4](✉), Parhat Abla[2], Zhiran Li[1,4], Lin Jiao[3], and Mingsheng Wang[1,4]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{chenhuiqin,liyongqiang,wangmingsheng}@iie.ac.cn
[2] School of Software, South China Normal University, Guangzhou, China
parhat@m.scnu.edu.cn
[3] State Key Laboratory of Cryptology, Beijing, China
[4] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. In this paper, we present quantum algorithms for finding impossible differentials and zero-correlation linear hulls, which are distinguishers for the two powerful attacks against symmetric ciphers of impossible differential attack and zero-correlation linear attack. Compared to classical methods, the proposed quantum algorithms possess many advantages. Firstly, our quantum algorithm for finding impossible differentials obtains the input and output differences by solving linear equation systems instead of searching in a limited space; Secondly, our quantum algorithm for zero-correlation linear hulls can investigate the key schedule's effect; Thirdly, the only computation cost of our algorithms is solving linear equation systems, and the size of the systems is not increasing as the round number increases.

The core idea of our method is to use the Berstein-Vazirani algorithm to find 1-linear structures of Boolean functions. We check the validity of the proposed quantum algorithm with the SIMON block cipher family and RC5 block cipher. We show that the proposed algorithms can discover some 11-round, 12-round, 13-round, 16-round, and 19-round impossible differentials and zero-correlation linear hulls of SIMON cipher when considering the key schedules and 2.5-round impossible differential of RC5 when considering the round subkeys.

Keywords: Quantum algorithm · Berstein-Vazirani algorithm · Linear structure · Impossible differential · Zero-correlation linear hull

1 Introduction

In recent years, there has been substantial research on quantum computers - machines that exploit quantum mechanical phenomena to solve mathematical

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 431–451, 2023.
https://doi.org/10.1007/978-3-031-35486-1_19

problems that are difficult or intractable for conventional computers. If large-scale quantum computers are ever built, they will threaten many of the cryptosystems currently in use, hence seriously compromising digital communications' confidentiality and integrity on the Internet and elsewhere.

As for symmetric cryptography, the most general result is Grover's algorithm [20], which can get an n-bit key with $\mathcal{O}(2^{n/2})$ quantum search. Then doubling the key length is a choice to resist this quantum exhaustive search attack. For example, the European PQCRYPTO project recommends using AES with 256-bit keys to provide 128-bit quantum security [14], and the 3GPP standardization organization requires increasing the security level to 256-bit key lengths for the 5G system [37]. With the rapid development of quantum technologies, it is essential to investigate more accurate cryptanalysis methods with quantum computation, giving a comprehensive understanding of the effectiveness of quantum computation in symmetric cryptography.

The most widely used algorithm is Simon's algorithm, which can find the unknown period of a Boolean function in linear time [10,40]. Many pioneer works apply Simon's algorithm in cryptography by using the hidden periods of various symmetric primitives. With this approach, Kuwakado and Morii first showed that the three-round Feistel network could be distinguished from random permutations [29]. Then they also proved that the Even-Mansour cipher could be broken [30]. Hosoyamada and Aoki [21] showed that the quantum-related key attack could also be applied in polynomial time with quantum computation. By combining Grover's algorithm and Simon's algorithm, Leander and May showed that FX-construction could be broken with the same time complexity as Grover's original algorithm [31]. Bonnetain et al. proposed offline Simon's algorithm, which improves significantly over the Grover-meets-Simon algorithm [6]. Recently, Bonnetain et al. broke many parallelizable MACs using Simon's algorithm and the new technique of quantum linearization [7].

Besides symmetric ciphers, the quantum computation can also affect the analysis of hash functions, MACs, and authenticated encryptions. Kaplan et al. showed that the most widely used modes of operation for authentication and authenticated encryption could be broken with Simon's algorithm [26]. Santoli and Schaffner also independently investigated this [38]. As for an n-bit hash function, Brassard et al. showed that collisions could be found with a query complexity of $\mathcal{O}(2^{n/3})$ with the BHT algorithm [11]. Hosoyamada and Sasaki gave more dedicated quantum collision attacks [23]. They showed that differential trails with a lower probability that cannot use in the classical setting might also lead to a collision attack in the quantum setting. Avoiding using large quantum random access memories (qRAMs), Dong et al. made an improvement by performing a quantum rebound attack based on differentials with non-full active super S-boxes to be searched with MILP-based method [17]. Recently, Hosoyamada and Sasaki have developed a dedicated quantum collision attack on SHA-256 and SHA-512, which can attack more steps than classical attacks [24].

In the present paper, we also aim to investigate the security evaluation of symmetric ciphers in the quantum setting. Unlike asymmetric algorithms, the security evaluation of symmetric ciphers relies on the resistance to known cryptanalysis methods, such as differential cryptanalysis [4], linear cryptanalysis [34], impossible differential cryptanalysis [3], zero-correlation cryptanalysis [5]. As for these attacks, the first step is to find a property that can distinguish the target cipher from random permutations. Such a property is usually called a distinguisher. For example, it needs to find an impossible differential or a zero-correlation linear hull before implementing an impossible differential or zero-correlation attack.

Nowadays, finding distinguishers of various cryptanalysis of symmetric ciphers classically has transitioned from manual analysis to automatic analysis based on the Mixed Integer Linear Programming (MILP) technique and the Boolean Satisfiability Problem (SAT) solvers. At the dawn of the quantum era, it is also of significant importance to investigate how to find distinguishers of symmetric ciphers with quantum computation. However, for the current known cryptanalysis methods of symmetric ciphers, there is a lack of research on finding their distinguishers quantumly, although there are already works diverted to present dedicated analysis. Kaplan et al. investigated the quantum version of differential attack and linear attack and gave some examples of applications on ciphers LAC and KLEIN [27]. Zhou et al. also gave the quantum version of the differential attack based on the quantum minimum/maximum-finding algorithm [45]. Hosoyamada and Sasaki showed that quantum Demirci and Selçuk (DS) meet-in-the-middle attacks could speed up significantly with quantum computers, and they applied this attack to 6-round generic Feistel constructions [22]. Using details of AES S-box differential property, Bonnetain et al. gave the first quantum DS-meet-in-the-middle attack on 8 rounds of AES-256 [8]. Bonnetain et al. further investigated quantum slide attacks [9]. It should note that besides attacks, some quantum distinguishers are also given in [9]. Dong et al. gave quantum advanced slide attacks [16]. Dunkelman et al. investigated quantum Time/Memory/Data tradeoff attacks and further improved Hellman's tradeoff curve [19]. Most of these works usually use quantum computation to reduce time complexity compared with classical analysis but do not focus on finding distinguishers. Especially, [15] investigate impossible differential attacks and improve the complexity of the sieving phase by the quantum technique.

Our Contributions. In the present paper, we present quantum algorithms for finding impossible differentials and zero-correlation linear hulls, which is the distinguisher for the impossible differential attack and zero-correlation linear attack. The core idea is to use a quantum algorithm to find linear structures of Boolean functions.

Note that the descriptions of the encryption algorithm, the decryption algorithm, and the key schedule algorithm are clearly known to attackers. With these descriptions, attackers can build the quantum circuits of a cipher's encryption, decryption, and key schedule in the quantum setting. We do not investigate the approach to give the quantum circuit of block ciphers in the present paper.

We use the quantum algorithm to find linear structures of Boolean functions and the "miss-in-the-middle" approach to find quantumly impossible differentials of symmetric ciphers. More precisely, we show that by finding 1-linear structures of the XOR of the i-th components of the r_1-round encryption (from the 0-th round to the r_1-th round) and r_2-round decryption (from the (r_1+r_2)-th round to the r_1-th round), one can obtain an (r_1+r_2)-round impossible differential. Compared to classical methods, the proposed quantum algorithm can investigate the details of all nonlinear functions and the effect of key schedules. Furthermore, the proposed quantum algorithm obtains an impossible differential's input and output differences by solving linear equations systems. In contrast, the input and output differences always need to be searched in a limited space for classical methods. Xie and Yang also proposed a quantum algorithm to find impossible differentials [44], and we show that our algorithm can find longer distinguishers than their method.

Furthermore, we prove that if there exists a vector c, such that it is an l-linear structure of the i-th component of r_1-round decryption (from the r_1-th round to the 0-th round), and it is also the $(l \oplus 1)$-linear structure of the j-th component of r_2-round encryption (from the r_1-th round to the (r_1+r_2)-th round), then (e_i, e_j) is an (r_1+r_2)-round zero-correlation linear hulls. On the basis of this result and the quantum algorithm for finding linear structures, we propose a quantum algorithm for finding zero-correlation linear hulls of symmetric ciphers. The quantum algorithm can investigate the details of all nonlinear functions. Moreover, for the first time, the effect of key schedules in finding distinguishers of zero-correlation linear attacks can be investigated.

To make it more clear, we present a comparison of our quantum algorithm and classical methods in Table 1.

Table 1. Comparisons with classical methods for finding IDs [a] /ZCs [b]

Methods	Features	Properties of Nonlinear functions	Data-dependent operations	Key Schedule	Problems need to solve	input and output difference	Ref.
$\mathcal{U}$-method, UID-method	IDs and ZCs	NO	NO	NO	/	Searching	[28,33]
$\mathcal{WW}$-method	IDs and ZCs	NO	NO	NO	linear equations system	Searching	[39,43]
MILP-based method	IDs and ZCs	small S-boxes ($\leq$ 6-bits)	NO	NO	optimize problems	Searching	[13]
SAT-based method	IDs and ZCs	big S-boxes (8,9-bits)	YES	YES	Nonlinear equations system	Searching	[25]
Quantum algorithm	IDs and ZCs	any nonlinear functions	YES	YES	linear equations system	Solving linear equations/ Searching	Sect.3 Sect.4

[a] IDs: Impossible Differentials [b] ZCs: Zero Correlation Linear Hulls

We check the validity of the proposed quantum algorithm with the SIMON block cipher family and the RC5 block cipher. We can verify that the proposed quantum algorithm can discover some 11-round, 12-round, 13-round, 16-round, and 19-round impossible differentials and zero-correlation linear hulls of SIMON32/64, SIMON48/72, SIMON64/96, SIMON96/96, SIMON128/128 when considering the key schedules, and 2.5-round impossible differentials of RC5 when considering the influence of round subkeys. The verification means that the round number of distinguishers of SIMON and RC5 found by quantum algorithms is at least equal to the round number of distinguishers of SIMON and RC5 found by classic methods. However, since we can not implement the proposed quantum algorithms in reality, we do not make sure whether they can find longer distinguishers than classic methods.

The paper is organized as follows. In Sect. 2, we give some preliminaries of the paper, including notations used, some basic results of linear structures of Boolean functions, the Bernstein-Vazirani Algorithm, and the quantum algorithm to find linear structures of Boolean functions. In Sect. 3, we give a quantum algorithm for finding impossible differentials and check the algorithm's validity with applications to SIMON and RC5. In Sect. 4, we give a quantum algorithm for finding zero-correlation linear hulls and check the algorithm's validity with applications to SIMON. A short conclusion is given in Sect. 5.

2 Preliminaries

2.1 Notations

The following notations are used in the present paper. Throughout the paper, we use $\oplus$ to denote the bitwise XOR of two vectors or XOR of two bits.

- $\mathbb{F}_2 = \{0, 1\}$: the finite field with 2 elements.
- $\mathbb{F}_2^n$: the vectors space over $\mathbb{F}_2$ with dimension n.
- $\mathcal{B}_n$: the set of all Boolean functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2$.
- e_i: the vector such that the i-th position equals 1, and other positions equal 0. For example, $e_1 = [1, 0, 0, \ldots, 0]$. Specially, $e_{i,j} = e_i \oplus e_j$.

2.2 Linear Structures of Boolean Functions

The following definitions and results are well-known in Boolean functions, and one can refer to [12] for a comprehensive survey of Boolean functions.

Definition 1. *We say $a \in \mathbb{F}_2^n$ is a linear structure of a Boolean function $f \in \mathcal{B}_n$, if*

$$f(x \oplus a) \oplus f(x) = f(a) \oplus f(0)$$

holds for all $x \in \mathbb{F}_n$.

Denote the set of all the linear structures of f as U_f. It is easy to see that U_f is a linear subspace of $\mathbb{F}_2^n$. More specifically, let U_f^i be the set of all linear structures with $f(a) \oplus f(0) = i$, that is, $U_f^i = \{a \in \mathbb{F}_2^n | f(x \oplus a) \oplus f(x) = i \text{ for all } x \in \mathbb{F}_2^n\}$, and U_f^i is usually called the set of i-linear structures of f, $i = 0, 1$. It is obvious that $U_f = U_f^0 \cup U_f^1$. Let $D_{f,a}^i = \{x \in \mathbb{F}_2^n | f(x \oplus a) \oplus f(x) = i\}$. Then it is easy to see that $a \in \mathbb{F}_2^n$ is an i-linear structure of f if and only if $D_{f,a}^i = \mathbb{F}_2^n$.

Definition 2. *For a Boolean function $f \in \mathcal{B}_n$, the Walsh spectrum is defined as*

$$S_f(\omega) = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus \omega \cdot x},$$

where $\omega \cdot x = \omega_1 x_1 \oplus \cdots \oplus \omega_n x_n$ is the usual inner product.

For the Walsh spectrum, Parseval's relation is well known.

Lemma 1 (Parseval's relation). *For a Boolean function $f \in \mathcal{B}_n$, we have $\sum\limits_{\omega \in \mathbb{F}_2^n} S_f^2(\omega) = 1$.*

Moreover, we introduce some results in the following, which play a vital role in the analysis of the linear structures of block ciphers using the Bernstein-Vazirani algorithm [2]. The proof of Lemma 2 and Lemma 3 can be obtained by consulting [18].

Lemma 2. *[18] For any $a \in \mathbb{F}_2^n$, $i \in \mathbb{F}_2$ and any Boolean function $f \in \mathcal{B}_n$, the following equality holds*

$$\sum_{\omega \in \mathbb{F}_2^n,\ \omega \cdot a = i} S_f^2(\omega) = \frac{|D_{f,a}^i|}{2^n},$$

where $|\cdot|$ means the number of elements in the set.

Lemma 3. *[18] For any Boolean function $f \in \mathcal{B}_n$, let $N_f = \{\omega \in \mathbb{F}_2^n \mid S_f(\omega) \neq 0\}$. Then, for any $i \in \mathbb{F}_2$, the set of the i-linear structure of f equals $\{a \in \mathbb{F}_2^n \mid \omega \cdot a = i,\ for\ all\ \omega \in N_f\}$.*

2.3 The Bernstein-Vazirani Algorithm

We do not introduce basic definitions of quantum computation here, and one can refer to the book [35] for fundamental knowledge. The Bernstein-Vazirani algorithm [2] can find the algebraic normal form of a linear Boolean function $f(x)$ with only one query on the superposition $\sum_{x \in \mathbb{F}_2^n} \frac{1}{\sqrt{2^n}} |x\rangle$. For a linear Boolean function $f(x) \in \mathcal{B}_n$, there exists exactly one $\alpha = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$, such that $f(x) = \alpha \cdot x = \sum_{i=1}^n a_i x_i$. Then the Bernstein-Vazirani algorithm can find $\alpha = (a_1, \ldots, a_n)$ with probability 1.

In the following, we give a brief introduction to the Bernstein-Vazirani algorithm. The illustration of the Bernstein-Vazirani algorithm is given in Fig. 1, where

- $|\psi_0\rangle = |0\rangle^n |1\rangle$;
- $|\psi_1\rangle = \sum_{x \in \mathbb{F}_2^n} \frac{|x\rangle}{\sqrt{2^n}} \frac{|0\rangle - |1\rangle}{\sqrt{2}}$;

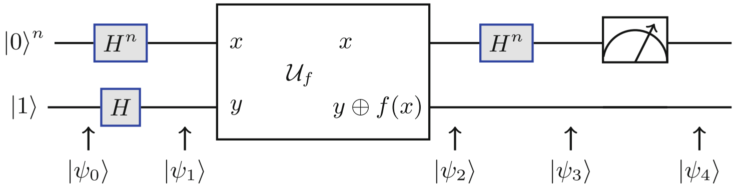


Fig. 1. Illustration of the Bernstein-Vazirani Algorithm

- $|\psi_2\rangle = \sum_{x\in\mathbb{F}_2^n} \frac{|x\rangle(-1)^{f(x)}}{\sqrt{2^n}} \frac{|0\rangle-|1\rangle}{\sqrt{2}}$;
- $|\psi_3\rangle = \sum_{z\in\mathbb{F}_2^n} (\sum_{x\in\mathbb{F}_2^n} \frac{(-1)^{f(x)\oplus x\cdot z}}{2^n}) \, |z\rangle \frac{|0\rangle-|1\rangle}{\sqrt{2}}$;
- $|\psi_4\rangle = \sum_{z\in\mathbb{F}_2^n} |\alpha\rangle \frac{|0\rangle-|1\rangle}{\sqrt{2}}$.

This means we can get the constant α with probability 1 after measuring the first n qubits of $|\psi_3\rangle$.

2.4 Finding Linear Structures of Boolean Functions with the Bernstein-Vazirani Algorithm

As shown in [32,44], one can use the Bernstein-Vazirani algorithm to find linear structures of Boolean functions. As for a Boolean function $f \in \mathcal{B}_n$, one can also apply the circuit of the Bernstein-Vazirani algorithm as above. Then the $|\psi_3\rangle$ can be rewritten as follows:

$$|\psi_3\rangle = \sum_{z\in\mathbb{F}_2^n} (S_f(z)) \, |z\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Hence, the measurement on the first n qubits will output a value z with probability $S_f^2(z)$, which means one can always obtain a value in N_f. Repetition of this procedure many times results in many elements of N_f, which contains a basis for the dual space of U_f^0 with extremely high probability. Note that U_f^0 is a subspace of $\mathbb{F}_2^n$ and $U_f^1 = \{b \oplus x \mid x \in U_f^0\}$ for some $b \in \mathbb{F}_2^n$. Then according to Lemma 3, one can get the set of i-linear structure of f by solving the linear system of $\omega \cdot a = i$ for all $\omega \in N_f$.

Theorem 1. [32,44] *Suppose $\alpha \in \mathbb{F}_2^n$ is an i-linear structure of a Boolean function $f \in \mathcal{B}_n$. Then Algorithm 1 returns α except a negligible probability.*

Algorithm 1: A quantum algorithm for finding linear structures of Boolean functions

```
The oracle access of $f(x)$ is given;
Let $p(n)$ be an arbitrary polynomial function of $n$, and initialize $H := \Phi$;
for $j = 1$ to $p(n)$ do
    Run Bernstein-Vazirani algorithm on $f$, and obtain a $n$-bit output
      $\omega = (\omega_1, ..., \omega_n) \in N_f$;
    Let $H = H \cup \{\omega\}$ ;
Let $\mathcal{S}^i = \{x \in \mathbb{F}_2^n \mid \omega \cdot x = i, \forall \omega \in H\}, i = 0, 1$;
if $\mathcal{S}^0 = \{0\}$ and $\mathcal{S}^1 = \emptyset$ then
    return "No";
else
    return $\mathcal{S}^0$ and $\mathcal{S}^1$;
```

3 A New Quantum Algorithm for Finding Impossible Differentials and Its Application to SIMON and RC5

Let $Enc_k(x)$ be an n-bit block cipher with primitive key $k \in \mathbb{F}_2^m$, and $Enc_k^r(x) = E_{k_r} \circ E_{k_{r-1}} \circ \cdots \circ E_{k_1}(x)$ be the r-round encryption of x under the primitive key k, where $k_i, 1 \le i \le r$ are round keys deriving by the key schedule algorithm under the primitive key k. Similarly, $Dec_k^r(x)$ denotes the corresponding decryption circuit of $Enc_k^r(x)$, that is, $Dec_k^r(Enc_k^r(x)) = x$. A pair $(\Delta x, \Delta y) \in \mathbb{F}_2^{2n}$, where $\Delta x \neq 0, \Delta y \neq 0$, is called an r-round impossible differential of $Enc_k(x)$ if

$$Enc_k^r(x \oplus \Delta x) \oplus Enc_k^r(x) = \Delta y$$

does not hold for any $x \in \mathbb{F}_2^n$ and $k \in \mathbb{F}_2^m$.

3.1 A New Quantum Algorithm for Finding Impossible Differentials

The Idea of Our Approach. We use the idea of "miss-in-the-middle", which is widely used to find impossible differential distinguishers of block ciphers. To get an impossible differential of $Enc_k^r(x)$, we divide it into the composition of r_1-th round encryption and r_2-th round encryption, i.e. $Enc_k^r(x) = Enc_k^{r_2} \circ Enc_k^{r_1}(x)$, where $r = r_1 + r_2$. Let $F_k(x, y) = Enc_k^{r_1}(x) \oplus Dec_k^{r_2}(y)$, and define the i-th component function of $F_k(x, y)$ as

$$f_i(x, y, k) = F_k[i](x, y) = Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y), 1 \le i \le n.$$

We call $(a, b, 0) \in \mathbb{F}_2^{2n}$ a non-trivial 1-linear structures of $f_i(x, y, k)$ if $(a, b, 0)$ is a 1-linear structure of $f_i(x, y, k)$ and $a \neq 0, b \neq 0$.

In the following result, we show that impossible differentials can be obtained by finding non-trivial 1-linear structures of $f_i(x, y, k)$.

Lemma 4. *Let $Enc_k(x)$ be a block cipher. If there exists $1 \leq i \leq n$, such that*

$$f_i(x, y, k) := Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y)$$

has a non-trivial 1-linear structure $(a, b, 0)$, then (a, b) is a (r_1+r_2)-round impossible differential of $Enc_k(x)$.

Proof. Suppose $(a, b, 0)$ is a non-trivial 1-linear structure of $f_i(x, y, k)$. Then $a \neq 0$ and $b \neq 0$, and we need to prove that the equation $Enc_k^r(x) \oplus Enc_k^r(x \oplus a) = b$ does not hold for any $x \in \mathbb{F}_2^n$ and $k \in \mathbb{F}_2^m$.

Note that $Enc_k^r(x) = Enc_k^{r_2} \circ Enc_k^{r_1}(x)$, then the above equation is equivalent to $Enc_k^{r_2} \circ Enc_k^{r_1}(x \oplus a) = Enc_k^{r_2} \circ Enc_k^{r_1}(x) \oplus b$. By composing $Dec_k^{r_2}(x)$, we get $Enc_k^{r_1}(x \oplus a) = Dec_k^{r_2}(Enc_k^{r_2} \circ Enc_k^{r_1}(x) \oplus b)$. Let $y = Enc_k^{r_2} \circ Enc_k^{r_1}(x)$. Then $Enc_k^{r_1}(x) = Dec_k^{r_2}(y)$ and hence we have

$$Enc_k^{r_1}(x \oplus a) \oplus Enc_k^{r_1}(x) = Dec_k^{r_2}(y \oplus b) \oplus Dec_k^{r_2}(y). \tag{1}$$

However, since $(a, b, 0)$ is a nontrivial 1-linear structure of $f_i(x, y, k)$, then

$$\begin{aligned} 1 &= f_i(x, y, k) \oplus f_i(x \oplus a, y \oplus b, k) \\ &= Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y) \oplus Enc_k^{r_1}[i](x \oplus a) \oplus Dec_k^{r_2}[i](y \oplus b) \end{aligned}$$

for all $x, y \in \mathbb{F}_2^n, k \in \mathbb{F}_2^m$. This means equality (1) does not hold for any $x, y \in \mathbb{F}_2^n$ and $k \in \mathbb{F}_2^m$. Therefore, (a, b) is an $(r_1 + r_2)$-round impossible differential of $Enc_k(x)$ and we complete the proof. □

Furthermore, 1-linear structures of $Enc_k[i](x) \oplus Dec_k[i](y), 1 \leq i \leq n$ can be found by Algorithm 1. More precisely, the quantum algorithm for finding impossible differentials is given in Algorithm 2.

Algorithm 2: A quantum algorithm to find impossible differentials

```
Given $r = r_1 + r_2$, and the quantum circuit of $Enc_k^{r_1}(x) \oplus Dec_k^{r_2}(y)$;
Let $p(n)$ be a polynomial function of $n$;
for $i = 1$ to $n$ do
    Let $H := \emptyset$;
    for $p = 1$ to $p(2n)$ do
        Run BV algorithm on $Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y)$ to get a
          $(2n+m)$-bit output
          $\omega = (w_1, \cdots, w_{2n}, \cdots, w_{2n+m}) \in N_{Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y)}$;
        Let $H = H \cup \{(w_1, \cdots, w_{2n})\}$;
    Solve the system of linear equations to get
      $\mathcal{S}^1 = \{(x, y) \in \mathbb{F}_2^{2n} \mid x \neq 0 \text{ and } y \neq 0 \text{ and } (x, y) \cdot \omega = 1, \forall \omega \in H\}$;
    if $\mathcal{S}^1 \neq \emptyset$ then
        return $\mathcal{S}_i^1$;
return "No";
```

About the Quantum Circuit of $Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y)$. Note that the description of the encryption algorithm, the decryption algorithm, and the key schedule are clearly known to attackers. Then an attacker can get the quantum circuit of the block cipher $Enc_k(x)$, and hence the quantum circuit of $Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y)$ for $1 \leq i \leq n$, where $x, y \in (\mathbb{F}_2^n)^2, k \in \mathbb{F}_2^m$ are all seen as variables.

About the Computing Complexity of Algorithm 2. Excepting the quantum circuit cost of $Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y)$, the only computing cost in Algorithm 2 is that the complexity of solving the system of linear equations

$$\begin{bmatrix} w_{1,1}, \cdots, w_{1,n}, w_{1,n+1}, \cdots, w_{1,2n} \\ w_{2,1}, \cdots, w_{2,n}, w_{2,n+1}, \cdots, w_{2,2n} \\ \vdots \\ w_{t,1}, \cdots, w_{t,n}, w_{t,n+1}, \cdots, w_{t,2n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

where $t = p(n)$ is the number of ω that we get after the inner iteration of Algorithm 2. The complexity of solving one system of linear equations as above is approximate $\mathcal{O}(8n^3)$, and we need to solve n different systems as above. Then the total computation complexity of finding r-round impossible differentials of a block cipher is about $\mathcal{O}(8rn^4)$ since we need to run Algorithm 2 for every pair $(r_1, r - r_1), 1 \leq r_1 \leq r - 1$.

About Success Probability. Algorithm 2 can find nontrivial 1-linear structures of $f_i(x, y, k) := Enc_k[i]^{r_1}(x) \oplus Dec_k^{r_2}[i](y)$. According to Theorem 1, if $f_i(x, y, k)$ has a nontrivial 1-linear structure, then by running Algorithm 2, we can get a nontrivial 1-linear structure except a negligible probability. The we have the following result.

Theorem 2. *Suppose $(a, b, 0)$ is a nontrivial 1-linear structure of $Enc_k[i]^{r_1}(x) \oplus Dec_k^{r_2}[i](y)$ for some $1 \leq i \leq n$. Then Algorithm 2 returns an r-round impossible differential (a, b) of the block cipher $Enc_k(x)$ except a negligible probability.*

Comparison with Classical Methods. In the classical setting, besides manual analysis, many methods studied how to get impossible differentials automatically, such as the $\mathcal{U}$-method [28], the UID-method [33], the $\mathcal{WW}$-method [43]. However, these methods can not consider the details of S-boxes. With the Mixed Integer Linear Programming (MILP) method, the difference property of small S-box and modular can be characterized, which means MILP-based methods can investigate small S-boxes' details and ARX ciphers [13,39]. However, the above methods are all based on the propagation of the differences and can not evaluate the effect of key schedules in the single-key setting. Hu et al. solved this problem by using the propagation of states and then solved the system of equations $Enc_k^r(x) \oplus Enc_k^r(x \oplus a) = b$ with SAT solvers directly [25].

We give comparisons with classical methods mentioned above in Table 1. Firstly, the biggest advantage of our quantum algorithm is that the input difference and output difference are obtained by solving linear equations systems.

In contrast, all classical methods need to search the input and output differences in limited spaces. Secondly, our quantum algorithm can also investigate the details of nonlinear functions of any size, investigate data-dependent operations, and investigate the effect of key schedules. Furthermore, our algorithm achieves all these advantages only with the computation cost of solving linear equations systems. Thus, our quantum algorithm surpasses all classical methods in the quantum setting.

Comparison with the Previous Quantum Algorithm. Xie and Yang presented an algorithm for identifying impossible differentials based on the quantum algorithm by finding linear structures of Boolean functions [44]. Their idea is directly applying Algorithm 1 to the components of $Enc_k^r(x)$. If a nontrivial i-linear structure a of the j-th component $Enc_k^r(x)$ is obtained, then it holds $Enc_k^r[j](x) \oplus Enc_k^r[j](x \oplus a) = i$ for all $x \in \mathbb{F}_2^n, k \in \mathbb{F}_2^m$. Therefore, for any vector $b \in \mathbb{F}_2^n$ with $b[j] = i \oplus 1$, $Enc_k^r[j](x) \oplus Enc_k^r[j](x \oplus a) = b$ does not hold for all x and k. Hence (a, b) is an r-round impossible differential.

It is intuitively that our approach finds longer round impossible differentials than Xie and Yang's approach since our approach is to find contradictions in the middle. For the block cipher $Enc_k(x)$, suppose r is the longest round of impossible differentials that finds with the algorithm in [44], i.e., we can find r-round i-linear structure of $Enc_k^r[j](x)$. Moreover, suppose the cipher $Enc_k(x)$ achieve full diffusion after r' rounds. Then we can find $(i \oplus 1)$-linear structures of $Dec_k^{r'-l}[j](x)$ for some $1 \leq l \leq r' - 1$ with high probability, and hence we can get a $(r + r' - l)$-round impossible differential of $Enc_k(x)$. Note that our algorithm can detect the later $(r + r' - l)$-round impossible differential, which means we can find the longer impossible differential distinguisher than Xie and Yang's method. In particular, we verify this by the application to the SIMON block cipher family in the present paper.

3.2 Application to SIMON

SIMON is a family of lightweight block ciphers designed by the NSA in 2013 [1]. It is a Feistel cipher with $2n$-bit state, where $n = 16, 24, 32, 48$ and 64. SIMON$2n/mn$ denotes a SIMON cipher with block size $2n$ bits and key size mn bits, where $m = 2, 3, 4$. In this section, we will show that by applying Algorithm 2 to SIMON, we can find the longest impossible differentials of SIMON in the quantum setting. Firstly, we give a method to check whether a given vector $c \in \mathbb{F}_2^n$ is a i-linear structure of a Boolean function $f(x) \in \mathcal{B}_n$ with SAT solvers, $i = 0, 1$.

About the Technique of Checking Linear Structures of Boolean Functions. We use the approach in [25] to check whether a given vector is a linear structure of a block cipher. For a given block cipher $Enc_k(x)$, a given vector $c \in \mathbb{F}_2^n$, the round number r and the output position j, we can model the following system of equations

$$Enc_k^r[j](x) \oplus Enc_k^r[j](x \oplus c) = i$$

in CVC format according to the propagation of states and the key schedule, and we save the corresponding statements as a file. Then we use STP, an openly available solver of the SMT problem, to solve the above problem. The STP will return "Valid" when the problem has no solution. Otherwise, it will return a solution to the problem. Thus if the above equation does not have solutions, then c is a $(i \oplus 1)$-linear structure of $Enc_k[j](x)$.

We have the following result by applying the technique of checking linear structures to SIMON32. The proof is given in Appendix A.

Proposition 1. *Suppose the round keys are random and independent. Then every component of the r-th round output of SIMON32 does not have linear structures for $r \geq 7$.*

The above result means that if we apply Xie and Yang's method to SIMON32, we can only find impossible differentials of SIMON32 as long as 6 rounds. However, the longest impossible differentials of SIMON32 is 11 rounds [42]. Therefore, the purpose of Algorithm 2 is to remedy the failure mentioned above.

Note that we can not apply Algorithm 2 directly in reality. Instead, we can verify that the longest impossible differentials of SIMON can be found by using Algorithm 2. With the method for checking linear structures of Boolean functions mentioned above, we can check that for some r-round impossible differentials (a, b) of SIMON, there exist r_1 and r_2 such that $(a, b, 0)$ is a non-trivial 1-linear structure of $Enc_k^{r_1}[j](x) \oplus Dec_k^{r_2}[j](y)$ for $1 \leq j \leq n$, $r_1 + r_2 = r$. Therefore, we can obtain (a, b) is an r-round impossible differential of SIMON using Algorithm 2 in the quantum setting.

SIMON32/64. The longest known impossible differentials of SIMON32 are 11 rounds. The 11-round impossible differentials (e_{32}, e_7), (e_{32}, e_9), $(e_{32}, e_{7,9})$ are applied in [42] to analysis SIMON32. We can check that

$$Enc_k^5[i](x) \oplus Enc_k^5[i](x \oplus e_{32}) \oplus Dec_k^6[i](y) \oplus Dec_k^6[i](y \oplus e_9) = 0$$

does not have solutions for $(x, y, k) \in \mathbb{F}_2^{2n+m}$ when considering the key schedule, where $i \in \{1, 8\}$. This means $(e_{32}, e_9, 0)$ is a non-trivial 1-linear structure of

$$Enc_k^5[i](x) \oplus Dec_k^6[i](y), i = 1, 8.$$

Besides, it can also check that $(e_{32}, e_{7,9}, 0)$ is a non-trivial 1-linear structure of $Enc_k^5[8](x) \oplus Dec_k^6[8](y)$ and $(e_{32}, e_7, 0)$ is a non-trivial 1-linear structure of $Enc_k^5[i](x) \oplus Dec_k^6[i](y), i = 8, 15$. Therefore, our quantum algorithm, i.e. Algorithm 2, can find the above impossible differentials.

In the same way, we can verify that the longest impossible differentials of SIMON48/72, SIMON64/96, SIMON96/96, and SIMON128/128 can be found using Algorithm 2, which is shown in Table 2.

3.3 Application to RC5

RC5, proposed by Rivest in 1994 [36], is an iterative cipher with $1 \leq r \leq 255$ ($r = 12$ being the value originally suggested). The design is based on a kind of

Table 2. The impossible differentials found by Algorithm 2 for SIMON

Ciphers	Previous results		Non-trivial 1-linear structure	
	DR	IDs	$(a, b, 0)$	$f_i(x, y, k)$
SIMON48/72	12	(e_{48}, e_1) $(e_{48}, e_{1,18})$ $(e_{48}, e_{1,22})$ $(e_{48}, e_{1,18,22})$	$(e_{48}, e_1, 0)$ $(e_{48}, e_{1,18}, 0)$ $(e_{48}, e_{1,22}, 0)$ $(e_{48}, e_{1,18,22}, 0)$	$Enc_k^6[1](x) \oplus Dec_k^6[1](y)$
SIMON64/96	13	(e_{33}, e_8)	$(e_{33}, e_8, 0)$	$Enc_k^7[32](x) \oplus Dec_k^6[32](y)$
SIMON96/96	16	(e_{94}, e_1)	$(e_{94}, e_1, 0)$	$Enc_k^9[47](x) \oplus Dec_k^7[47](y)$
SIMON128/128	19	(e_{65}, e_2)	$(e_{65}, e_2, 0)$	$Enc_k^{11}[2](x) \oplus Dec_k^8[2](y)$

-DR: The longest impossible differentials that we have known for the cipher.
-IDs: The input and output difference of the known impossible differential.
-$f_i(x, y, k) := Enc_k^{r_1}[i](x) \oplus Dec_k^{r_2}[i](y)$, which is defined in Lemma 4.

Feistel network, where one round can be split into two Feistel-like halves, each of which consists of XOR, variable rotation, and modular addition.

Previous Results. Since the variable rotation operation of RC5 highly depends on the value of the state, it cannot be handled by the previous automatic search model until an automatic search tool for the impossible differentials by considering the propagation of states was proposed in [25]. In their work, they proved that the longest impossible differentials of RC5 is 2.5-round and listed 12 impossible differentials for RC5-32.

Our Results. According to the longest known impossible differentials of RC5, the 2.5-round impossible differentials $(e_{i,i+16}, e_{17}), 1 \leq i \leq 12$ are applied to analysis RC5-32. Similarly to the technique in Sect. 3.2, we can check that

$$Enc_k^1[j](x) \oplus Enc_k^1[j](x \oplus e_{i,i+16}) \oplus Dec_k^{1.5}[j](y) \oplus Dec_k^{1.5}[j](y \oplus e_{17}) = 0$$

does not have solutions for $(x, y, k) \in \mathbb{F}_2^{2n+m}$ when considering the influence of round subkeys, where $1 \leq i \leq 12$ and $1 \leq j \leq 16$. For example, it means that $(e_{i,i+16}, e_{17}, 0)$ is a non-trivial 1-linear structure of

$$Enc_k^1[j](x) \oplus Dec_k^{1.5}[j](y), 1 \leq j \leq 16,$$

when $(e_{12,28}, e_{17})$ is an impossible differential of RC5. Therefore, we can find the longest impossible differentials of RC5 by applying Algorithm 2.

4 A Quantum Algorithm for Finding Zero-Correlation Linear Hulls and Its Application to SIMON

Zero-correlation analysis is introduced by Bogdanov and Rijmen in [5]. For the round function E of the block cipher $Enc_k(x)$, a linear correlation of E, which is denoted by $C_E(u, v)$, is defined as $C_E(u, v) = S_{v \cdot E}(u) = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot E(x) \oplus u \cdot x}$. A linear trail $U = (u_0, u_1, \ldots, u_r)$ is the concatenation of r correlation of round function, and the correlation of linear trail is defined as $C_U = \prod_{i=1}^{r} C_E(u_{i-1}, u_i)$. For $\alpha, \beta \in \mathbb{F}_2^n$, the linear hull of $Enc_k^r(x)$ is defined as

$$C(\alpha, \beta) = \sum_{U: u_0 = \alpha, u_r = \beta} C_U.$$

(α, β) is called an r-round zero-correlation linear hull of $Enc_k(x)$ if $C(\alpha, \beta) = 0$.

4.1 A Quantum Algorithm to Find Zero-Correlation Linear Hulls

In this subsection, we propose a quantum algorithm for finding zero-correlation linear hulls of symmetric ciphers. Firstly, we have the following result. which is helpful for identifying zero-correlation linear hulls with the quantum algorithm.

Lemma 5. *Let $F(x) = F_2(F_1(x))$. If $U^i_{F_2[j_2]} \cap U^{i\oplus 1}_{F_1^{-1}[j_1]} \neq \emptyset$ for some $i \in \mathbb{F}_2$, then (e_{j_1}, e_{j_2}) is a zero-correlation linear hull of $F(x)$.*

Proof. Suppose that there exist $i \in \mathbb{F}_2$ and $0 \le j_2, j_1 \le n-1$ such that $U^i_{F_2[j_2]} \cap U^{i\oplus 1}_{F_1^{-1}[j_1]} \neq \emptyset$. Let $c \in U^i_{F_2[j_2]} \cap U^{i\oplus 1}_{F_1^{-1}[j_1]}$. Then $F_2[j_2](x) \oplus F_2[j_2](x \oplus c) = i$ and $F_1^{-1}[j_1](x) \oplus F_1^{-1}[j_1](x \oplus c) = i \oplus 1$. According to Lemma 2, we have

$$\sum_{\omega \cdot c = i} S^2_{F_2[j_2]}(\omega) = \frac{|D^i_{F_2[j_2],c}|}{2^n} = 1, \text{ and } \sum_{\omega \cdot c = i\oplus 1} S^2_{F_1^{-1}[j_1]}(\omega) = \frac{|D^{i\oplus 1}_{F_1^{-1}[j_1],c}|}{2^n} = 1.$$

Then according to Parseval's relation (Lemma 1), it holds that

$$\sum_{\omega \cdot c = i\oplus 1} S^2_{F_2[j_2]}(\omega) = 0, \text{ and } \sum_{\omega \cdot c = i} S^2_{F_1^{-1}[j_1]}(\omega) = 0,$$

from which we get that $C_{F_2}(\omega, e_{j_2}) = 0$ when $c \cdot \omega = i \oplus 1$, and $C_{F_1^{-1}}(\omega, e_{j_1}) = 0$ when $c \cdot \omega = i$. Furthermore, note that

$$C_{F_1^{-1}}(\omega, e_{j_1}) = \sum_{x \in \mathbb{F}_2^n} (-1)^{e_{j_1} \cdot F_1^{-1}(x) \oplus \omega \cdot x} = \sum_{x \in \mathbb{F}_2^n} (-1)^{\omega \cdot F_1(x) \oplus e_{j_1} \cdot x} = C_{F_1}(e_{j_1}, \omega).$$

Then for the linear hull (e_{j_2}, e_{j_1}) of $F(x)$, it can be computed as

$$\begin{aligned} C(e_{j_1}, e_{j_2}) &= \sum_{\omega \in \mathbb{F}_2^n} C_{F_1}(e_{j_1}, \omega) C_{F_2}(\omega, e_{j_2}) = \sum_{\omega \in \mathbb{F}_2^n} C_{F_1^{-1}}(\omega, e_{j_1}) C_{F_2}(\omega, e_{j_2}) \\ &= \sum_{\omega \in \mathbb{F}_2^n : c \cdot \omega = i} C_{F_1^{-1}}(\omega, e_{j_1}) C_{F_2}(\omega, e_{j_2}) + \sum_{\omega \in \mathbb{F}_2^n : c \cdot \omega = i \oplus 1} C_{F_1^{-1}}(\omega, e_{j_1}) C_{F_2}(\omega, e_{j_2}) \\ &= \sum_{\omega \in \mathbb{F}_2^n : c \cdot \omega = i} 0 \times C_{F_2}(\omega, e_{j_2}) + \sum_{\omega \in \mathbb{F}_2^n : c \cdot \omega = i \oplus 1} C_{F_1^{-1}}(\omega, e_{j_1}) \times 0 \\ &= 0. \end{aligned}$$

This means that (e_{j_2}, e_{j_1}) is a zero-correlation linear hull of $F(x) = F_2(F_1(x))$, and we complete the proof. □

The Idea of our Algorithm. Let $Enc_k(x)$ be a symmetric cipher, and let r, r_1, r_2 be integers with $r = r_1 + r_2$. The idea of finding r-round zero-correlation linear hulls of $Enc_k(x)$ is as follows. Writing $Enc_k^r(x) = Enc_k^{r_2} \circ Enc_k^{r_1}(x)$, similar to Algorithm 1, using the Bernstein-Vazirani algorithm, we can find linear structure sets $U^l_{Dec_k^{r_1}[i](x)}$ and $U^{l\oplus 1}_{Enc_k^{r_2}[j](x)}$ for each $1 \leq i, j \leq n$ respectively. If

$$\exists\, c \in U^l_{Dec_k^{r_1}[i](x)} \cap U^{l\oplus 1}_{Enc_k^{r_2}[j](x)}$$

for some $l \in \mathbb{F}_2$, then according to Lemma 5, we can get an r-round zero-correlation of $Enc_k(x)$. Our quantum algorithm for finding zero-correlation linear hulls is given in Algorithm 3.

Similarly to Theorem 2, it is easy to get Theorem 3 according to Theorem 1 and Lemma 5.

Algorithm 3: A quantum algorithm for zero-correlation linear hulls

```
Given $r = r_1 + r_2$, the oracle access of $Enc_k^{r_2}(x_1, \ldots, x_n)$, $Dec_k^{r_1}(y_1, \ldots, y_n)$;
Let $p(n)$ be an arbitrary polynomial function of $n$;
for $j_1 = 1, 2, \ldots, n$ do
    Let $H_1 = \emptyset$;
    for $i = 1, 2, \ldots, p(n)$ do
        Run BV algorithm on $Enc_k^{r_2}[j_1](x)$, and obtain a $(n+m)$-bit output
          $\omega \in N_{Enc_k^{r_2}[j_1]}$;
        Let $H_1 = H_1 \cup \{(\omega_1, \cdots, \omega_n)\}$;
    Solve the systems of linear equations to get
      $\mathcal{S}_{j_1}^0 = \{x \in \mathbb{F}_2^n \mid x \cdot \omega = 0,\ \forall \omega \in H_1\}$;
      $\mathcal{S}_{j_1}^1 = \{x \in \mathbb{F}_2^n \mid x \cdot \omega = 1,\ \forall \omega \in H_1\}$;
    if $\mathcal{S}_{j_1}^l \neq \emptyset$ for some $l \in \{0, 1\}$ then
        for $j_2 = 1, 2, \ldots, n$ do
            set $H_2 := \emptyset$;
            for $p = 1$ to $p(n)$ do
                Run BV algorithm on $Dec_k^{r_1}[j_2](x)$ to get a $(n+m)$-bit ourput
                  $\omega \in N_{Dec_k^{r_1}[j_2]}$;
                Let $H_2 = H_2 \cup \{(\omega_1, \cdots, \omega_n)\}$;
            Solve the systems of linear equations to get
              $\mathcal{S}_{j_2}^{l\oplus 1} = \{x \in \mathbb{F}_2^n \mid x \cdot \omega = l \oplus 1,\ \forall \omega \in H_2\}$;
            if $\mathcal{S}_{j_1}^l \cap \mathcal{S}_{j_2} \neq \emptyset$ then
                Return $(e_{j_1}, e_{j_2})$
Return "No" ;
```

Theorem 3. *Suppose $U^l_{Dec_k^{r_1}[j_1](x)} \cap U^{l\oplus 1}_{Enc_k^{r_2}[j_2](x)} = \emptyset$. Then Algorithm 3 returns an r-round zero-correlation linear hull (e_{j_1}, e_{j_2}) of the block cipher $Enc_k^r(x)$ except a negligible probability.*

Proof. Let $c \in U^{l}_{Dec_k^{r_1}[j_1](x)} \cap U^{l\oplus 1}_{Enc_k^{r_2}[j_2](x)}$. Then according to Lemma 5, (e_{j_1}, e_{j_2}) is a zero-correlation linear hull of $E_k^{r_1+r_2}(x)$. According to Theorem 1, the vector c can by found in $U^{l}_{Dec_k^{r_1}[j_1](x)}$ and $U_{Enc_k^{r_2}[j_2](x)}$ except a negligible probability. Thus Algorithm 3 returns (e_{j_1}, e_{j_2}) except a negligible probability. □

Comparison with the Classical Methods. First of all, the quantum algorithm for finding zero-correlation linear hulls can investigate the details of nonlinear functions with any size and the effect of the key schedule. Second, the quantum algorithm can achieve these advantages with the computational cost of solving linear equation systems. Furthermore, the size of linear equations systems does not increase as the round number increase. While in the classical setting, the problems that need to be solved are becoming more and more complex with the increase of round numbers. The comparison with the classical methods mentioned above is shown in Table 1. Therefore, the quantum algorithm also surpasses classical methods.

4.2 Application to SIMON

Apply the above approach to the SIMON block cipher family. We can verify that some longest known zero-correlation distinguishers can be obtained with the quantum algorithm.

SIMON32/64. The longest known zero-correlation linear hull of SIMON32 is 11 rounds. In [41,42], it is shown that (e_{16}, e_{25}) is a 11-round zero-correlation linear hull of SIMON32. We can check that for $c = e_8$, $Enc_k^5[25](x) \oplus Enc_k^5[25](x \oplus c) = 0$, and $Dec_k^6[16](x) \oplus Dec_k^6[16](x \oplus c) = 1$ holds for all $x \in \mathbb{F}_2^{32}$ and $k \in \mathbb{F}_2^{64}$, which means

$$c = e_8 \in U^1_{Dec_k^6[16](x)} \cap U^0_{Enc_k^5[25](x)}.$$

Furthermore, we also have

$$e_{24} \in U^1_{Dec_k^5[16](x)} \cap U^0_{Enc_k^6[25](x)},$$

$$e_1 \in U^0_{Dec_k^6[16](x)} \cap U^1_{Enc_k^5[25](x)},\ e_{17} \in U^0_{Dec_k^5[16](x)} \cap U^1_{Enc_k^6[25](x)}.$$

Then by running Algorithm 3, we can obtain that (e_{16}, e_{25}) is a zero-correlation linear hull of SIMON32/64 when considering the key schedule.

In the same way, we can verify that the longest zero-correlation linear hulls of SIMON48/72, SIMON64/96, SIMON96/96, and SIMON128/128 can be found using Algorithm 3, which is shown in Table 3.

Table 3. The zero-correlation linear hulls found by Algorithm 3 for SIMON

Ciphers	Previous results		$U^i_{F_2[j_2]} \cap U^{i\oplus 1}_{F_1^{-1}[j_1]}$
	DR	ZCLHs	
SIMON48/72	12	(e_{24}, e_{47})	$c = e_{47} \in U^0_{Dec_k^6[24](x)} \cap U^1_{Enc_k^6[47](x)}$
SIMON64/96	13	(e_{32}, e_{57})	$c = e_1 \in U^0_{Dec_k^8[32](x)} \cap U^1_{Enc_k^5[57](x)}$ $c = e_{33} \in U^0_{Dec_k^7[32](x)} \cap U^1_{Enc_k^6[57](x)}.$
SIMON96/96	16	(e_{48}, e_{93})	$c = e_{47} \in U^0_{Dec_k^{10}[48](x)} \cap U^1_{Enc_k^6[93](x)}$ $c = e_{95} \in U^0_{Dec_k^9[48](x)} \cap U^1_{Enc_k^7[93](x)}.$
SIMON128/128	19	(e_{64}, e_{127})	$c = e_{63} \in U^0_{Dec_k^{12}[64](x)} \cap U^1_{Enc_k^7[127](x)}$ $c = e_{127} \in U^0_{Dec_k^{11}[64](x)} \cap U^1_{Enc_k^8[127](x)}$ $c = e_{64} \in U^1_{Dec_k^8[64](x)} \cap U^0_{Enc_k^{11}[127](x)}$ $c = e_{128} \in U^1_{Dec_k^7[64](x)} \cap U^0_{Enc_k^{12}[127](x)}$

- ZCLHs: The input and output mask of the known zero-correlation linear hulls.

-$U^i_{F_2[j_2]} \cap U^{i\oplus 1}_{F_1^{-1}[j_1]}$: According to Lemma 5, if $c \in U^i_{F_2[j_2]} \cap U^{i\oplus 1}_{F_1^{-1}[j_1]}$ for some $i \in \mathbb{F}_2$, then (e_{j_1}, e_{j_2}) is a zero-correlation linear hull of $F(x)$.

5 Conclusion

In the present paper, we propose new quantum algorithms for finding impossible differentials and zero-correlation linear hulls of block ciphers. Compared to classical methods, the only computational cost of the quantum algorithm is the solution of linear equation systems, and the size of the linear equation systems does not increase as the number of rounds increases. Furthermore, the proposed quantum algorithm obtains the input and output differences of an impossible differential by solving linear equations systems instead of searching in limited spaces like classical methods.

We provide the SIMON block cipher family and RC5 as examples in the paper to verify our quantum algorithm's validity and effectiveness. We show that the quantum algorithms can find some longest known impossible differentials and zero-correlation linear hulls of SIMON block cipher when considering the key schedules. In addition, we verify the 2.5-round impossible differential of RC5 when considering the key round subkeys with our quantum algorithm. Other symmetric ciphers can also be applied and tried later. Our original intention is to develop a good quantum-based tool for finding distinguishers of symmetric ciphers that can be used when a large number of the spread of quantum computers are achieved. We will continue to follow up and hope that more quantum algorithms will be developed to find other known, or newly even better, cryptanalysis methods' distinguisher. This will be very helpful for the design and security evaluation of symmetric ciphers in quantum time.

Acknowledgements. The authors are grateful to the anonymous reviewers for their insightful comments. This work was supported partly by the National Key Research and Development Program of China (Grant No. 2022YFF0604702).

A Proof of Proposition 1

Proof. We prove this based on the propagation of difference. Remember that the round function of SIMON is $F(x) = ((S^1(x)) \wedge (S^8(x))) \oplus (S^2(x))$, and one round transformation is

$$(L_i, R_i) \mapsto (R_i \oplus F(L_i) \oplus K_i, L_i),$$

where K_i is the i-th round key.

For an input difference $a \in \mathbb{F}_2^{16}$ of SIMON32, we denote the output difference set as $\Delta_F(a) = \{F(x) \oplus F(x \oplus a) \mid x \in \mathbb{F}_2^{16}\}$. For $1 \leq i \leq 16$, if $v[i]$ is a constant for all $v \in \Delta_F(a)$, then we call i a determinant position of $\Delta_F(a)$. Otherwise, i is called an indeterminant position of $\Delta_F(a)$, and we use the variable X to denote the value. Then $\Delta_F(a)$ can be represented by a vector. For example,

$$\Delta_F(e_1) = [0, 0, 0, 0, 0, 0, 0, 0, X, 0, 0, 0, 0, 0, 1, X].$$

Note that for $i \in \mathbb{F}_2$, we always have $X \oplus i = X$, $X \wedge i = X$. Then for an input difference vector a of $F(x)$, we can get the output difference vector $\Delta_F(a)$ according to the following difference equation

$$F(x) \oplus F(x \oplus a) = S^1(a) \wedge S^8(x) \oplus S^8(a) \wedge S^1(x) \oplus F(a).$$

Thus for a given pair $(a_0, b_0) \in \mathbb{F}_2^{32}$, we can compute (a_{i+1}, b_{i+1}) for $i \geq 0$ recursively via

$$(a_{i+1}, b_{i+1}) := (\Delta_F(a_i) \oplus b_i, a_i),$$

until all positions in (a_{i+1}, b_{i+1}) become X.

At last, for $a_0 \in \mathbb{F}_2^{16}$, we do not need to search all $b_0 \in \mathbb{F}_2^{16}$. Note that $(a_1, b_1) = (\Delta_F(a_0) \oplus b_0, a_0)$, and $X \oplus i = X$ for $i \in \mathbb{F}_2$. Then we only need to search b_0 in the following set

$$S_{a_0}^{b_0} := \{v \mid v[i] \in \mathbb{F}_2 \text{ for } i \in S_{a_0}^c; v[i] := X \text{ for } i \in S_{a_0}^X\},$$

where $S_{a_0}^c = \{1 \leq i \leq 16 \mid i \text{ is a determinant position of } \Delta_F(a_0)\}$, and $S_{a_0}^X = \{1 \leq i \leq 16 \mid i \text{ is a un-determinant position of } \Delta_F(a_0)\}$. This means if $\Delta_F(a_0)$ has k determinant positions, then we only need to search 2^k b_0, which reduce the search space greatly.

With the above approach, we search all $a_0 \in \mathbb{F}_2^{16}$ and $b_0 \in S_{a_0}^{b_0}$. There are (a_0, b_0) such that the 6-round output has determinant positions, which means that there exists $1 \leq i \leq 32$, such that the i-th component of the 6-round output of SIMON32 has linear structures. However, for all $a_0 \in \mathbb{F}_2^{16}$ and $b_0 \in S_{a_0}^{b_0}$, every position of the corresponding 7-round output becomes X. This result means all the 7-round output components of SIMN32 do not have linear structures. Then we complete the proof. □

References

1. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. cryptology eprint archive (2013)
2. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM J. Comput. **26**(5), 1411–1473 (1997). https://doi.org/10.1137/S0097539796300921
3. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. J. Cryptol. **18**(4), 291–311 (2005). https://doi.org/10.1007/s00145-005-0129-3
4. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991). https://doi.org/10.1007/BF00630563
5. Bogdanov, A., Rijmen, V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Des. Codes Crypt. **70**(3), 369–383 (2012). https://doi.org/10.1007/s10623-012-9697-z
6. Bonnetain, X., Hosoyamada, A., Naya-Plasencia, M., Sasaki, Yu., Schrottenloher, A.: Quantum attacks without superposition queries: the offline Simon's algorithm. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 552–583. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_20
7. Bonnetain, X., Leurent, G., Naya-Plasencia, M., Schrottenloher, A.: Quantum linearization attacks. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13090, pp. 422–452. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92062-3_15
8. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Trans. Symmetric Cryptol. **2019**(2), 55–93 (2019). https://doi.org/10.13154/tosc.v2019.i2.55-93
9. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: On quantum slide attacks. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 492–519. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38471-5_20
10. Brassard, G., Hoyer, P.: An exact quantum polynomial-time algorithm for Simon's problem. In: Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems, pp. 12–23. IEEE (1997). https://doi.org/10.1109/ISTCS.1997.595153
11. Brassard, G., HØyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 163–169. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054319
12. Carlet, C.: Boolean Functions for Cryptography and Coding Theory. Cambridge University Press, Cambridge (2021). https://doi.org/10.1017/9781108606806
13. Cui, T., Chen, S., Jia, K., Fu, K., Wang, M.: New automatic search tool for impossible differentials and zero-correlation linear approximations. Cryptology ePrint Archive (2016). http://eprint.iacr.org/2016/689
14. Daniel, A., Lejla, B., et al.: Initial recommendations of long-term secure post-quantum systems. PQCRYPTO. EU. Horizon **2020** (2015)
15. David, N., Naya-Plasencia, M.: Quantum impossible differential attack. Applications to CLEFIA, AES and SKINNY. Master's thesis, MPRI (2019). https://hal.inria.fr/hal-02424410
16. Dong, X., Dong, B., Wang, X.: Quantum attacks on some Feistel block ciphers. Des. Codes Crypt. **88**(6), 1179–1203 (2020). https://doi.org/10.1007/s10623-020-00741-y
17. 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. LNCS, vol. 12492, pp. 727–757. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_25

18. Dubuc, S.: Characterization of linear structures. Des. Codes Crypt. **22**(1), 33–45 (2001)
19. Dunkelman, O., Keller, N., Ronen, E., Shamir, A.: Quantum time/memory/data tradeoff attacks. Cryptology ePrint Archive, Report 2021/1561 (2021). https://ia.cr/2021/1561
20. 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). https://doi.org/10.1145/237814.237866
21. Hosoyamada, A., Aoki, K.: On quantum related-key attacks on iterated Even-Mansour ciphers. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **102**(1), 27–34 (2019). https://doi.org/10.1587/transfun.E102.A.27
22. Hosoyamada, A., Sasaki, Yu.: Quantum demiric-selçuk meet-in-the-middle attacks: applications to 6-round generic Feistel constructions. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 386–403. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_21
23. Hosoyamada, A., Sasaki, Yu.: Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, Part II, vol. 12106, pp. 249–279. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_9
24. Hosoyamada, A., Sasaki, Yu.: Quantum collision attacks on reduced SHA-256 and SHA-512. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 616–646. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_22
25. Hu, X., Li, Y., Jiao, L., Tian, S., Wang, M.: Mind the propagation of states. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 415–445. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_14
26. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, Part II, vol. 9815, pp. 207–237. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_8
27. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. IACR Trans. Symmetric Cryptol. **2016**(1), 71–94 (2016). https://doi.org/10.13154/tosc.v2016.i1.71-94
28. Kim, J., Hong, S., Sung, J., Lee, S., Lim, J., Sung, S.: Impossible differential cryptanalysis for block cipher structures. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 82–96. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-24582-7_6
29. Kuwakado, H., Morii, M.: Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In: 2010 IEEE International Symposium on Information Theory, pp. 2682–2685. IEEE (2010). https://doi.org/10.1109/ISIT.2010.5513654
30. Kuwakado, H., Morii, M.: Security on the quantum-type Even-Mansour cipher. In: 2012 International Symposium on Information Theory and its Applications, pp. 312–316. IEEE (2012)
31. Leander, G., May, A.: Grover meets Simon – quantumly attacking the FX-construction. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 161–178. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_6

32. Li, H., Yang, L.: A quantum algorithm to approximate the linear structures of Boolean functions. Math. Struct. Comput. Sci. **28**(1), 1–13 (2018). https://doi.org/10.1017/S0960129516000013
33. Luo, Y., Lai, X., Wu, Z., Gong, G.: A unified method for finding impossible differentials of block cipher structures. Inf. Sci. **263**, 211–220 (2014). https://doi.org/10.1016/j.ins.2013.08.051
34. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_33
35. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, Cambridge (2010). https://doi.org/10.1017/CBO9780511976667
36. Rivest, R.L.: The RC5 encryption algorithm. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 86–96. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60590-8_7
37. SA3, G.: Tr 33.841 study on supporting 256-bit algorithms for 5g (2018). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3422
38. Santoli, T., Schaffner, C.: Using Simon's algorithm to attack symmetric-key cryptographic primitives (2017). arXiv. 1603.07856, https://doi.org/10.26421/QIC17.1-2-4
39. Sasaki, Yu., Todo, Y.: New Impossible Differential Search Tool from Design and Cryptanalysis Aspects. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, Part III, vol. 10212, pp. 185–215. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_7
40. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474–1483 (1997)
41. Sun, L., Fu, K., Wang, M.: Improved zero-correlation cryptanalysis on SIMON. In: Lin, D., Wang, X.F., Yung, M. (eds.) Inscrypt 2015. LNCS, vol. 9589, pp. 125–143. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38898-4_8
42. Wang, Q., Liu, Z., Varıcı, K., Sasaki, Yu., Rijmen, V., Todo, Y.: Cryptanalysis of reduced-round SIMON32 and SIMON48. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 143–160. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13039-2_9
43. Wu, S., Wang, M.: Automatic search of truncated impossible differentials for word-oriented block ciphers. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 283–302. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_17
44. Xie, H., Yang, L.: Using Bernstein–Vazirani algorithm to attack block ciphers. Des. Codes Crypt. **87**(5), 1161–1182 (2018). https://doi.org/10.1007/s10623-018-0510-5
45. Zhou, Q., Lu, S., Zhang, Z., Sun, J.: Quantum differential cryptanalysis. Quant. Inf. Proc. **14**, 2101–2109 (2015). https://doi.org/10.1007/s11416-021-00395-x

Memory-Efficient Quantum Information Set Decoding Algorithm

Naoto Kimura[1], Atsushi Takayasu[1,2](✉), and Tsuyoshi Takagi[1]

[1] Graduate School of Information Science and Technology,
The University of Tokyo, Tokyo, Japan
{kimura-naoto077,takayasu-a,takagi}@g.ecc.u-tokyo.ac.jp
[2] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Abstract. Code-based cryptography is a candidate for post-quantum cryptography and the security of code-based cryptosystems relate to the hardness of the syndrome decoding problem. The Information Set Decoding (ISD) algorithm initiated by Prange is a typical method for solving the syndrome decoding problem. Various methods have been proposed that make use of exponentially large lists to accelerate the ISD algorithm. Furthermore, Bernstein (PQCrypto 2010) and Kachigar and Tillich (PQCrypto 2017) applied Grover's algorithm and quantum walks to obtain *quantum* ISD algorithms that are much faster than their classical ones. These quantum ISD algorithms also require exponentially large lists as the classical algorithms, and they must be kept in quantum states. In this paper, we propose a new quantum ISD algorithm by combining Both and May's classical ISD algorithm (PQcrypto 2018), Grover's algorithm, and Kirshanova's quantum walk (PQCrypto 2018). The proposed algorithm keeps an exponentially large list in the quantum state just like the existing quantum ISD algorithms, but the list size is much smaller. Although the proposed algorithm is slower than the existing algorithms when there is sufficient quantum memory, it is fastest when the amount of quantum memory is limited. Due to the property, we believe that our algorithm will be the fastest ISD algorithm in actual quantum computing since large-scale quantum computers seem hard to realize.

Keywords: code-based cryptography · syndrome decoding problem · information set decoding · quantum algorithm

1 Introduction

1.1 Background

Due to the invention of Shor's quantum algorithm [27] RSA and elliptic curve cryptosystems can be broken in polynomial time by a quantum computer. It is an urgent issue to develop practical post-quantum cryptographic schemes before

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 452–468, 2023.
https://doi.org/10.1007/978-3-031-35486-1_20

a large-scale general-purpose quantum computer is realized. Based on the situation, NIST is already working on the post-quantum cryptography standardization project [23].

Code-based cryptography (e.g., [21,22]) is a candidate of post-quantum cryptography. Indeed, Classic McElice, BIKE, and HQC are listed as Round 4 Candidates of the NIST standardization project [24]. The security of code-based schemes relates to the computational hardness of the *syndrome decoding problem*. Here, the syndrome decoding problem is that given a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, syndrome vector $\mathbf{s} \in \mathbb{F}_2^{n-k}$, and integer w, the solution is a vector $\mathbf{e} \in \mathbb{F}_2^n$ that satisfies $\mathbf{He} = \mathbf{s}$ and the Hamming weight of $\mathbf{e}$ is w. In order to make practical use of code-based schemes, it is necessary to properly evaluate the hardness of the syndrome decoding problem. In 1962, Prange [25] proposed the *Information Set Decoding* (ISD) algorithm for solving the syndrome decoding problem. Prange's ISD algorithm (`Prange`) can solve the syndrome decoding problem in time exponential in n and the space complexity polynomial in n. After the proposal of `Prange`, faster ISD algorithms have been proposed by Stern (`Stern`) [28], May et al. (`MMT`) [19], Becker et al. (`BJMM`) [2], May and Ozerov (`MO`) [20], and Both and May[1] (`BM17` [5] and `BM18` [4,6]). These algorithms accelerate `Prange` by storing exponentially many vectors in lists. Therefore, as opposed to `Prange`, the space complexities of these faster variants are exponential in n. For example, `Prange` runs in time[2] $\widetilde{\mathcal{O}}(2^{0.1207n})$, while the `BJMM` algorithm runs in time $\widetilde{\mathcal{O}}(2^{0.1020n})$ and space $\widetilde{\mathcal{O}}(2^{0.0728n})$. Moreover, `MO`, `BM17`, and `BM18` are faster than `BJMM`. The fastest `BM18` runs in time $\widetilde{\mathcal{O}}(2^{0.0885n})$ and space $\widetilde{\mathcal{O}}(2^{0.0736n})$. Although the exponential memory may become a bottleneck in practice [7], these improvement should be theoretically interesting. Esser and Bellini [10] proposed a syndrome decoding simulator to estimate the running time of known ISD algorithms with limited memory.

Furthermore, it is known that these "classical" ISD algorithms are combined with quantum algorithms such as Grover's algorithm [14] and quantum walks to obtain "quantum" ISD algorithms. Bernstein [3] proposed a quantum `Prange` that runs in time $\widetilde{\mathcal{O}}(2^{0.060350n})$. Similarly, Kachigar and Tillich [15] proposed a quantum ISD algorithm for `Stern`, `MMT`, and `BJMM`. Among them, quantum `BJMM` which is the fastest quantum ISD algorithm runs in time $\widetilde{\mathcal{O}}(2^{0.058660n})$ and space $\widetilde{\mathcal{O}}(2^{0.023413n})$. Kirshanova [16] claimed that the classical `BM17`, which is faster than the classical `BJMM`, is hard to get quantum speed-up.

Compared with the case of *classical* ISD algorithms, exponential memory is a critical problem for *quantum* ISD algorithms. Indeed, the number of qubits in the current state-of-the-art physical realization of quantum computers is still ≈ 100. To understand the situation more accurately, we briefly recall the practical realization of Shor's algorithm. Shor's algorithm is an efficient factorizing algorithm since it can factor composite numbers $N = pq$ in time $\mathcal{O}(\log^3 N)$. Moreover,

[1] See also [9,11] as the follow-up works of the algorithms.

[2] We note that the syndrome decoding problem which we study in this paper is full distance decoding.

the algorithm requires only $\approx 2\log N$ qubits. In contrast, the largest composite number on which physical implementations of Shor's algorithm worked is $21 = 3 \times 7$ [1]. The authors [1] also reported that factoring $35 = 5 \times 7$ is still infeasible. Although the number of qubits is not the only bottleneck of this experiment, quantum computers we will have in the future may only be able to handle the small number of qubits. Therefore, taking into account the practical implementation possibilities on quantum computers, the improvement of quantum ISD algorithms with limited space complexity should be an important research topic. Esser et al. [12] studied the problem and proposed a hybrid ISD algorithm; however, the work only focuses on `Prange`.

1.2 Our Contribution

In this paper, we propose a quantum ISD algorithm that is a quantum variant of `BM18` [6] by combining with Grover's algorithm [14] and Kirshanova's quantum walk [16]. The algorithm runs in time $\widetilde{\mathcal{O}}(2^{0.059809n})$ and space $\widetilde{\mathcal{O}}(2^{0.0014901n})$. Unfortunately, the proposed algorithm is slower than quantum `BJMM`. On the other hand, the proposed algorithm requires much less space. We analyze the minimum time complexities of quantum `Prange`, `Stern`, `MMT`, `BJMM` when the amount of space complexity is limited. Then, the quantum `BJMM` which is the fastest among known quantum ISD algorithms runs in time $\widetilde{\mathcal{O}}(2^{0.060027n})$ if it can use space only $\widetilde{\mathcal{O}}(2^{0.0014901n})$. Moreover, we find that the proposed algorithm is the fastest when the space complexity is bounded by $\widetilde{\mathcal{O}}(2^{0.0028250n})$.

1.3 Overview

Due to the explanation so far, keen readers may have the following two questions:

(1) Why do our proposed algorithm is faster than the other quantum ISD algorithms only when the space complexity is limited although the *classical* `BM18` is faster than the other classical ISD algorithms without the restrictions?
(2) How can we construct a quantum variant of `BM18` although Kirshanova claimed that `BM18` seems difficult to get quantum speed-up?

The high-level answer is that the proposed algorithm is not a *fully* quantized variant of `BM18`, but its *partially* quantized variant. Most ISD algorithms start by preparing 2^d lists and search the solution of the syndrome decoding problem, where each list is a set of vectors. In each step with $2^{d-d'}$ lists of the depth d', the algorithm updates them to $2^{d-d'-1}$ lists of the depth $d'+1$. Finally, the algorithm searches the solution by combining two lists with the depth $d-1$. Briefly speaking, the improvements of classical ISD algorithms so far are due to the analysis of the number of depth d and the speed-up of updating lists in each step. `BM18` sets the depth $d = 4$ and makes use of `MO`'s nearest neighbor search [20] to accelerate the update.

Kirshanova [16] proposed a quantum walk algorithm for the nearest neighbor search. Thus, it seems that we can construct a quantum variant of `BM18`. However, the approach becomes less efficient if the depth is large such as $d = 4$ of BM18 due to the property of Kirshanova's quantum walk. To avoid the issue, we construct a *partially* quantized variant of BM18. Our proposed algorithm uses four lists of the depth $d = 2$[3]. To update the four lists to two lists, we do not use Kirshanova's quantum walk but update them classically. We use Kirshanova's quantum walk only when we search for the solution of the syndrome decoding problem from the two lists. In other words, our proposed algorithm does not make use of the full advantage of BM18. That is why the proposed algorithm improved the other quantum ISD algorithms only when the space complexity is limited. In contrast, since we use Kirshanova's quantum walk only for depth one, we can avoid the concern claimed by Kirshanova [16] and achieve the quantum speed-up.

1.4 Roadmap

In Sects. 2 and 3, we review the overviews of the *classical* and *quantum* ISD algorithms, respectively. In Sect. 4, we propose our quantum ISD algorithm. In Sect. 5, we compare our proposed ISD algorithm and the other quantum ISD algorithms.

Notation. $\mathbb{F}_2$ represents a finite field with an order of 2. Let lowercase bold letters (e.g., $\mathbf{e}$) and uppercase bold letters (e.g., $\mathbf{H}$) denote column vectors and matrices, respectively. Let $\mathbf{e}^\top$ and $\mathbf{H}^\top$ denote the transposes of $\mathbf{e}$ and $\mathbf{H}$, respectively. Let $\mathbf{0}_n$ denote an n-dimensional zero vector. Let $\mathbf{I}_m$ denote an identity matrix of the size $m \times m$. For two matrices $\mathbf{H}_1 \in \mathbf{F}_2^{m \times n_1}$ and $\mathbf{H}_2 \in \mathbf{F}_2^{m \times n_2}$, let $(\mathbf{H}_1 \mid \mathbf{A}_2) \in \mathbf{F}_2^{m \times (n_1 + n_2)}$ denote their concatenation. For a vector $\mathbf{e}$, let $\mathrm{wt}(\mathbf{e})$ denote the Hamming weight of $\mathbf{e}$. For two vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ with the same dimension, let $\mathrm{dt}(\mathbf{e}_1, \mathbf{e}_2)$ denote the Hamming distance between $\mathbf{e}_1$ and $\mathbf{e}_2$. For a function $f(n)$, we use the notation $\widetilde{\mathcal{O}}(f(n))) = \mathrm{O}(f(n) \cdot (\log_2 f(n))^m)$ for a constant m. For a finite set $\mathcal{L}$, let $|\mathcal{L}|$ denote the number of elements in $\mathcal{L}$ and let $a \overset{\$}{\leftarrow} \mathcal{L}$ denote a sampling of a from $\mathcal{L}$ uniformly at random. For a real number x such that $0 \le x \le 1$, let $H_2(x) \coloneqq -x \log_2 x - (1 - x) \log_2(1 - x)$ denote the binary entropy function. Throughout the paper, we set $0 \log_2 0 = 0$. For a real vector $\mathbf{p} \in [0, 1]^m$ such that $\sum_{i=1}^m p_i = 1$, let $H(\mathbf{p}) \coloneqq -\sum_{i=1}^m p_i \log_2 p_i$ denote the entropy function.

2 Classical Information Sed Decoding Algorithm

In this section, we review the *classical* ISD algorithms. We first review the definition of the syndrome decoding problem. Then, we review the original ISD algorithm Prange and summarize the known techniques to accelerate Prange.

[3] The depth $d = 2$ is not an optimized value. If we use the larger d, we may be able to obtain the faster quantum ISD algorithms. However, we cannot analyze the time complexity of larger $d \ge 3$ since the numerical analysis of the computational complexity requires huge time due to more parameters that should be optimized.

2.1 Syndrome Decoding Problem

We first review the notion of a linear code, parity check matrix, and syndrome. The k-dimensional subspace $\mathcal{C}$ on $\mathbb{F}_2^n$ with the minimum distance $d := \min\{\mathrm{dt}(\mathbf{x},\mathbf{y}) \mid \mathbf{x},\mathbf{y} \in \mathcal{C}, \mathbf{x} \neq \mathbf{y}\}$ is called a binary $[n,k,d]$-linear code. For a binary $[n,k,d]$-linear code, there is a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ satisfying $\mathcal{C} = \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathbf{Hx} = \mathbf{0}_{n-k}\}$. For a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ and vector $\mathbf{x} \in \mathbb{F}_2^n$, $\mathbf{s} = \mathbf{Hx} \in \mathbb{F}_2^{n-k}$ is called a syndrome of $\mathbf{x}$. The task of the syndrome decoding problem with $(\mathbf{H},\mathbf{s},n,k,w)$ is given a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$, and integer $w \in \mathbb{N}$, and finding a vector $\mathbf{e} \in \mathbb{F}_2^n$ satisfying

$$\mathbf{s} = \mathbf{He} \quad \text{and} \quad \mathrm{wt}(\mathbf{e}) = w.$$

In this paper, we assume that there is at least one solution. Moreover, we study the case when $w = d \leq n \cdot {H_2}^{-1}(1 - k/n)$ holds called *full distance decoding*, where ${H_2}^{-1}(\cdot)$ is the inverse function of $H_2(\cdot)$. Here, we obtain the value by the Gilbert-Varshamov bound $k/n = 1 - H_2(w/n)$ for $n \to \infty$.

The time and space complexity of the ISD algorithm are expressed as $\widetilde{\mathcal{O}}(2^{tn})$ and $\widetilde{\mathcal{O}}(2^{sn})$, respectively, with two constants t and s by ignoring $\mathsf{poly}(n)$ factors. If the space complexity is polynomial in n, we simply write $s = 0$. The complexity of the ISD algorithm is analyzed in the average case. We use $T(\mathbf{\Theta}_{\mathrm{ISD}})$ to denote the time complexity of the ISD algorithm, where $\mathbf{\Theta}_{\mathrm{ISD}}$ is the set of all parameters of the ISD algorithm other than n,k,w. Here, all the parameters in $\mathbf{\Theta}_{\mathrm{ISD}}$ are independent of k. To compare each ISD algorithm, the time complexity is determined by $\max_{k,w} \min_{\mathbf{\Theta}_{\mathrm{ISD}}} T(\mathbf{\Theta}_{\mathrm{ISD}})$, where the space complexity is analyzed by using the same k, w, and $\mathbf{\Theta}_{\mathrm{ISD}}$.

2.2 Basic Information Set Decoding Algorithm: `Prange`

We provide an overview of `Prange`. To be precise, the description and analysis are based on Lee and Brickell [17]. Given $(\mathbf{H},\mathbf{s},n,k,w)$, `Prange` first samples a permutation matrix $\mathbf{P} \xleftarrow{\$} \mathbb{F}_2^{n\times n}$, then finds a vector $\tilde{\mathbf{e}} \in \mathbb{F}_2^n$ so that $\mathbf{P}\tilde{\mathbf{e}}$ is the solution of the syndrome decoding problem. When we can find a solution with the permutation matrix $\mathbf{P}$, we say that $\mathbf{P}$ is *appropriate*. Specifically, `Prange` with a parameter[4] $p \in \{0,\ldots,w\}$ consists of the following two steps:

(Step 1: Randomizing a Parity Check Matrix) Sample a permutation matrix $\mathbf{P} \xleftarrow{\$} \mathbb{F}_2^{n\times n}$ and set

$$\widetilde{\mathbf{H}} := \mathbf{HP}.$$

If there is no singular matrix $\mathbf{T} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ such that

$$\mathbf{T}\widetilde{\mathbf{H}} = (\widetilde{\mathbf{Q}} \mid \mathbf{I}_{n-k})$$

[4] To be precise, the original `Prange` fixes the parameter to $p = 0$.

with some matrix $\widetilde{\mathbf{Q}} \in \mathbb{F}_2^{(n-k)\times k}$, start the Step 1 from the beginning. Otherwise, set $\tilde{\mathbf{s}} = \mathbf{Ts}$.

(Step 2: Finding a Solution) For all $\binom{k}{p}$ vectors $\tilde{\mathbf{e}}' \in \mathbb{F}_2^k$ such that $\mathrm{wt}(\tilde{\mathbf{e}}') = p$, compute

$$\tilde{\mathbf{e}}'' = \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}' + \tilde{\mathbf{s}} = \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}' + \mathbf{Ts}.$$

If there is no vector $\tilde{\mathbf{e}}''$ satisfying $\mathrm{wt}(\tilde{\mathbf{e}}'') = w-p$, go back to Step 1. Otherwise, output

$$\mathbf{e} = \mathbf{P}(\tilde{\mathbf{e}}' \mid \tilde{\mathbf{e}}'').$$

Correctness. Here, we check that the vector $\mathbf{e}$ output by `Prange` is a correct solution. Due to the definitions and conditions of $\widetilde{\mathbf{H}}, \mathbf{T}, \tilde{\mathbf{e}}$, and $\tilde{\mathbf{e}}''$, it holds that

$$\mathbf{THe} = \mathbf{T}\widetilde{\mathbf{H}}(\tilde{\mathbf{e}}' \mid \tilde{\mathbf{e}}'') = (\widetilde{\mathbf{Q}} \mid \mathbf{I}_{n-k})(\tilde{\mathbf{e}}' \mid \tilde{\mathbf{e}}'') = \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}' + \tilde{\mathbf{e}}'' = \mathbf{Ts}.$$

Since $\mathbf{T}$ is non-singular, it holds that

$$\widetilde{\mathbf{H}}(\tilde{\mathbf{e}}' \mid \tilde{\mathbf{e}}'') = \mathbf{H}(\mathbf{P}(\tilde{\mathbf{e}}' \mid \tilde{\mathbf{e}}'')) = \mathbf{s}.$$

Thanks to the fact that $\mathbf{P}$ is a permutation matrix and the conditions of the Hamming weight $\mathrm{wt}(\tilde{\mathbf{e}}') = p$ and $\mathrm{wt}(\tilde{\mathbf{e}}'') = w - p$ in Step 2, it holds that

$$\mathrm{wt}(\mathbf{e}) = \mathrm{wt}(\tilde{\mathbf{e}}') + \mathrm{wt}(\tilde{\mathbf{e}}'') = w.$$

Therefore, the vector $\mathbf{e}$ is the correct solution of the syndrome decoding problem.

Computational Cost. The average computational cost of `Prange` is a product of the expected number of iterations for sampling an appropriate permutation matrix $\mathbf{P}$ and the computational cost of each iteration. The computational cost of Step 1 is polynomial and that of Step 2 is $\widetilde{\mathcal{O}}\left(\binom{k}{p}\right)$ that denotes the number of vectors $\tilde{\mathbf{e}}'$ satisfying $\mathrm{wt}(\tilde{\mathbf{e}}') = p$. Next, we analyze the expected number of iterations. Observe that $\mathbf{P}^{-1}$ is a permutation matrix satisfying $(\tilde{\mathbf{e}}' \mid \tilde{\mathbf{e}}'') = \mathbf{P}^{-1}\mathbf{e}$. When we assume that $\tilde{\mathbf{e}}'$ and $\tilde{\mathbf{e}}''$ are uniformly random vectors in $\mathbb{F}_2^k$ and $\mathbb{F}_2^{n-k}$ satisfying $\mathrm{wt}(\tilde{\mathbf{e}}') = p$ and $\mathrm{wt}(\tilde{\mathbf{e}}'') = w-p$, respectively, there are $\binom{k}{p}\binom{n-k}{w-p}$ appropriate matrices $\mathbf{P}$. In contrast, there are $\binom{n}{w}$ vectors $\tilde{\mathbf{e}}$ that are permutations of $\mathbf{e}$ such that $\mathrm{wt}(\mathbf{e}) = w$. Thus, the probability $\mathcal{P}(p)$ for sampling an appropriate permutation matrix $\mathbf{P}$ under the parameter p is $\mathcal{P}(p) = \binom{k}{p}\binom{n-k}{w-p}\binom{n}{w}^{-1}$. In other words, the expected number of iterations for sampling an appropriate permutation matrix $\mathbf{P}$ is $\mathcal{P}(p)^{-1}$. Therefore, the average computational cost of `Prange` is $\widetilde{\mathcal{O}}\left(\mathcal{P}(p)^{-1} \cdot \binom{k}{p}\right)$. When we use the notations $c_k := k/n$, $c_w := w/n$, and $c_p := p/n$ and the approximation $\binom{n}{r} = \widetilde{\mathcal{O}}(2^{n \cdot H_2(r/n)})$, the average computational cost of `Prange` is

$$\widetilde{\mathcal{O}}\left(\mathcal{P}(p)^{-1} \cdot \binom{k}{p}\right) = \widetilde{\mathcal{O}}\left(\binom{n-k}{w-p}^{-1} \cdot \binom{n}{w}\right) = \widetilde{\mathcal{O}}\left(2^{t_{\mathtt{Prange}}(p) \cdot n}\right),$$

where

$$t_{\mathtt{Prange}} := H_2(c_w) - (1-c_k)H_2\left(\frac{c_w - c_p}{1-c_k}\right).$$

Finally, we numerically optimize the parameters c_k, c_w, c_p and obtain $\max_k \min_p t_{\mathtt{Prange}}(0) = 0.1207$.

2.3 Advanced Information Set Decoding Algorithms

Among several techniques to improve `Prange`, we briefly review `Stern`'s technique, `MMT`'s representation technique, and `MO`'s nearest neighbor search that are used by `BM18`.

`Stern` 's Technique. As we observed above, the computational complexity of the ISD algorithm is expressed as the product of the number of iterations and the search time. `Stern` [28] (and its improved variants [8,13]) which may be combined with [26] used a list storing exponentially many vectors and speeds up the ISD algorithm by increasing the number of iterations but significantly reducing the search time of each iteration. The technique has also been used by subsequent works with some modifications. As a brief explanation, `Stern` first samples a permutation matrix[5] $\mathbf{P}$ in Step 1 and searches for $\tilde{\mathbf{e}}'$ as in `Prange`. Unlike `Prange`, instead of searching for $\tilde{\mathbf{e}}'$ from the whole space $\mathbb{F}_2^k$, `Stern` introduced additional constraints of the Hamming weights on $\tilde{\mathbf{e}}'_1$ and $\tilde{\mathbf{e}}'_2$ and reduce the search space of $\tilde{\mathbf{e}}'$. As a result, `Stern` runs in time $\widetilde{\mathcal{O}}(2^{0.1166n})$ that is faster than `Prange`. In contrast, `Stern` requires a list of the size $\widetilde{\mathcal{O}}(2^{0.0312n})$ to search for $\tilde{\mathbf{e}}'$ efficiently.

Representation Technique. `MMT` [19] is based on `Stern` and divide $\tilde{\mathbf{e}}'$ into four sub-vectors. `MMT` runs in time $\widetilde{\mathcal{O}}(2^{0.1116n})$ with a list of the size $\widetilde{\mathcal{O}}(2^{0.0541n})$. `BJMM` [2] is also based on `Stern` and divide $\tilde{\mathbf{e}}'$ into eight sub-vectors. Furthermore, `BJMM` used a representation technique that increases the search space of $\tilde{\mathbf{e}}'$ and reduces the number of iterations. For a vector $\mathbf{v} \in \mathbb{F}^n$ satisfying $\mathrm{wt}(\mathbf{v}) = w$, we call a pair of vectors $(\mathbf{v}_1, \mathbf{v}_2) \in (\mathbb{F}_2^n)^2$ a representation of $\mathbf{v}$ if $\mathbf{v}_1$ and $\mathbf{v}_2$, where the right sub-vector of $\mathbf{v}_1$ and left sub-vector of $\mathbf{v}_2$ are zero vectors, satisfy $\mathrm{wt}(\mathbf{v}_1) = \mathrm{wt}(\mathbf{v}_2) = w' \geq w/2$ and $\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2$. See Fig. 1 that illustrates the overview of the representation. There are $\binom{w}{w/2}\binom{n-w}{w'-w/2}$ representations $(\mathbf{v}_1, \mathbf{v}_2)$ of $\mathbf{v}$ satisfying $\mathrm{wt}(\mathbf{v}) = w$. The number of representations are larger than the search space with the simple division $\mathbf{v} = (\mathbf{v}_1 \mid \mathbf{v}_2)$. As a result, `BJMM` runs in time $\widetilde{\mathcal{O}}(2^{0.1020n})$ with a list of the size $\widetilde{\mathcal{O}}(2^{0.0738n})$.

Nearest Neighbor Search. `BM18` [4,6] utilized `MO`'s γ-nearest neighbor search algorithm [20] that solves the following problem.

Definition 1 (γ-Nearest Neighbor Search [4,20]). *Given two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ with elements taken uniformly at random from $\mathbb{F}_2^n$ and satisfying $|\mathcal{L}_1|, |\mathcal{L}_2| \leq 2^{\lambda n}$. Let ℓ be an integer such that $1 \leq \ell \leq n$. For a fixed integer $0 \leq \ell' \leq n - \ell$,*

[5] To be precise, the condition of $\mathbf{P}$ is not the same as that of `Prange`.

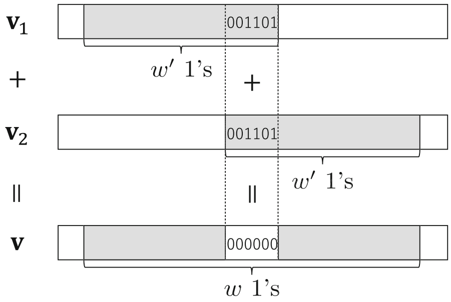


Fig. 1. $(\mathbf{v}_1, \mathbf{v}_2)$ which is the representation of $\mathbf{v}$

let $(\mathbf{x})_\ell = (x_{\ell'+1}, \ldots, x_{\ell'+\ell}) \in \mathbb{F}_2^\ell$ *denote a sub-vector of* $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}_2^n$*. The task of the* (γ, ℓ)*-NN problem is finding all but negligible fraction of the pairs* $(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ *satisfying* $\mathrm{dt}((\mathbf{x}_1)_\ell, (\mathbf{x}_2)_\ell) \leq \gamma n$ *for some* $0 \leq \gamma \leq n/2$*.*

MO's γ-nearest neighbor search algorithm[6] can solve the problem in average time $\widetilde{\mathcal{O}}(2^{n \cdot \tau_{\mathrm{NN}}(\lambda, \delta, \gamma)})$. Here, it holds that

$$\tau_{\mathrm{NN}}(\lambda, \delta, \gamma) = \tau\left(\frac{\lambda}{\delta}, \frac{\gamma}{\delta}\right) \cdot \delta$$

if it holds that

$$\frac{1}{2} \cdot \tau\left(\frac{\lambda}{\delta}, \frac{\gamma}{\delta}\right) < \frac{\lambda}{\delta} < 1 - H_2\left(\frac{\gamma}{2\delta}\right), \quad \tau_{\mathrm{NN}}(\lambda, \delta, \gamma) = \min\left\{\tau_1, \tau_2\right\},$$

where

$$\delta := \frac{\ell}{n}, \quad \tau(\lambda, \gamma) := (1-\gamma)\left(1 - H_2\left(\frac{{H_2}^{-1}(1-\lambda) - \gamma/2}{1-\gamma}\right)\right),$$
$$\tau_1 = 2\lambda, \quad \tau_2 = \max\left\{H_2\left(\frac{\gamma}{2\delta}\right)\delta + \lambda, 2\lambda + H_2\left(\frac{\gamma}{\delta}\right)\delta - \delta\right\}.$$

Thanks to the MO's algorithm, BM18 runs in time $\widetilde{\mathcal{O}}(2^{0.0982n})$ with a list of the size $\widetilde{\mathcal{O}}(2^{0.0717n})$.

3 Quantum Information Set Decoding Algorithm

In this section, we summarize an overview of the quantum ISD algorithms. We first review Grover's algorithm [14] and the quantum walk [16,18]. Then, we review the quantum ISD algorithm and summarize the computational complexity.

[6] To be precise, MO's proposed γ-nearest neighbor search algorithm [20] does not work for arbitrary parameters. In such cases, we use other algorithms for solving the problem. See [4] for the detail.

Grover's Algorithm. Let E be a finite set. Given a function $f : E \to \{0,1\}$, Grover's algorithm [14] finds $x \in E$ satisfying $f(x) = 1$. Let ε denote a proportion of $x \in E$ satisfying $f(x) = 1$. Let T_f denote a time for computing $f(x)$. Grover's algorithm finds a solution in average time $\mathrm{O}(T_f/\sqrt{\varepsilon})$.

Quantum Walk. Let V and E denote sets of vertices and edges of a d-regular undirected connected graph $G = (V, E)$, respectively. Here, we say that an undirected graph G is d-regular if all vertices of G has the same degree d. The quantum walk is given $G = (V, E)$ and $M \subset V$, then finding $x \in M$. Let $T_{\mathtt{S}}$, $T_{\mathtt{U}}$, and $T_{\mathtt{C}}$ denote the time for constructing G, updating the step of a random walk, and checking whether $x \in M$ holds, respectively. Let ε denote a proportion of $x \in V$ satisfying $x \in M$. Let $\mathbf{A}$ denote an adjacency matrix of G and δ denote a spectral gap of $\mathbf{M} := (1/d)\mathbf{A}$, where $\mathbf{M}$ denotes a probability matrix in the Markov chain due to the equal probability of a point on G moving to other adjacent vertices. Here, the spectral gap is $\delta := 1 - \max_{\lambda \neq 1} |\lambda|$, where λ is an eigenvalue of $\mathbf{M}$. The quantum walk runs in time $\mathcal{O}\left(T_{\mathtt{S}} + 1/\sqrt{\varepsilon} \cdot \left((1/\sqrt{\delta})F \cdot T_{\mathtt{U}} + T_{\mathtt{C}}\right)\right)$, where $\delta > 0$ if the Markov chain is acyclic.

Next, we review the notion of a Johnson Graph.

Definition 2 (Johnson Graph). *Let n and r denote integers satisfying $r \leq n$. For a list L such that $|L| = n$, we call graph $J(n, r) = (V, E)$ satisfying the following three conditions a Johnson graph:*

- *$J(n, r)$ is undirected.*
- *Vertices of $J(n, r)$ are subsets of L of the size r. In other words, it holds that $V = \{S \subset L : |S| = r\}$.*
- *There is an edge between two vertices S and S' iff $|S \cap S'| = r - 1$, i.e., $E = \{(S, S') : |S \cap S'| = r - 1\}$.*

Since Johnson graph $J(n, r)$ is an $n(n-r)$-regular undirected connected graph, the quantum walk is applicable and a spectral gap of the graph is $n/(r(n-r))$.

Next, we explain the direct product of the Johnson graph.

Definition 3 (Direct Product of Johnson Graphs). *Let n, r be natural numbers satisfying $r \leq n$, and suppose that we are given two* Johnson *graphs $J(n, r) = (V, E)$ and $J'(n, r) = (V', E')$. In this case, the direct product $J(n, r) \times J'(n, r)$ is determined by the following (1) and (2) (the same applies to the direct product of three or more Johnson graphs).*

(1) *The vertex S is the source of $V \times V'$. That is, the vertex set is $\{(v_1, v_2) : (v_1, v_2) \in V \times V'\}$.*
(2) *Let $(S_1, S_2), (S_1', S_2') \in V \times V'$, $S := (S_1, S_2)$,$S' := (S_1', S_2')$. S, S' is connected by an edge if and only if it satisfies "$S_1 = S_1'$ and $S' \in E'$" or "$S_2 = S_2'$ and $S \in E$". That is, the edge set is $\{(S, S') : (S_1 = S_1' \wedge S' \in E') \vee (S_2 = S_2' \wedge S \in E)\}$.*

The direct product of k-Johnson graphs $J_1(n, r) \times \cdots \times J_k(n, r)$ is a connected undirected graph that is $r(n-r)$-regular, and its spectral gap δ satisfies $\delta \leq n/(kr(n-r))$.

Finally, we describe Kirshanova's quantum walk [16] and its computational complexity. The algorithm is given the lists $\mathcal{L}_1, \mathcal{L}_2 \subset \mathbb{F}_2^n$ and $\gamma \in [0, 1/2]$, then check whether there exists $\mathbf{v}_1 \in \mathcal{L}_1, \mathbf{v}_2 \in \mathcal{L}_2$ satisfying $\mathrm{dt}(\mathbf{v}_1, \mathbf{v}_2) = \gamma n$. The average time and space computations of this algorithm are both expressed as $\widetilde{\mathcal{O}}(2^{(1-H(\mathbf{p}(\alpha,\beta,\gamma)+H(\mathbf{q}(\gamma))\cdot n)})$ [16]. Here, $\mathbf{p}(\alpha, \beta, \gamma)$ and $\mathbf{q}(\gamma)$ are elements in $[0, 1]^8$ and $[0, 1]^4$, respectively. Let a_i and b_i denote i-th elements of $\mathbf{p}(\alpha, \beta, \gamma)$ and $\mathbf{p}(\gamma)$, respectively, where

$$a_1 = a_2 = \frac{1}{2} - \frac{\gamma + \beta + \alpha}{4}, \quad a_3 = a_4 = \frac{\gamma + \beta - \alpha}{4},$$
$$a_5 = a_6 = \frac{\gamma + \alpha - \beta}{4}, \quad a_7 = a_8 = \frac{\beta + \alpha - \gamma}{4},$$
$$b_1 = b_2 = \frac{1 - \gamma}{2}, \quad b_3 = b_4 = \frac{\gamma}{2}.$$

Here, $\alpha, \beta \in [0, 1]$ are parameters such that $\alpha = H^{-1}(1 - \log_2 r)$, $\beta = H^{-1}(1 - (3/4)\log_2 r)$ by setting the parameter of Johnson graph $\mathcal{L}_1, \mathcal{L}_2$ as $r = |\mathcal{L}_1 \cup \mathcal{L}_2|^{\frac{2}{3}}$.

Quantum ISD Algorithm. The quantum walk and the ISD algorithm are related by the k-list partial sum problem. The k-list subproblem is that, given k-lists $\mathcal{L}_1, \ldots, \mathcal{L}_k$ on $\mathbb{F}_2^n$ with equal number of elements and a function $g : \mathcal{L}_1 \times \ldots \times \mathcal{L}_k \to \{0, 1\}$, finding $(\mathbf{v}_1, \ldots, \mathbf{v}_k) \in \mathcal{L}_1 \times \cdots \times \mathcal{L}_k$ such that $g(\mathbf{v}_1, \ldots, \mathbf{v}_k) = 1$. The existing ISD algorithms after `Prange` reduce the amount of time computation by solving the k-list partial sum problem. By performing a quantum walk on the direct product of the Johnson graph $J_1(n, r) \times \cdots \times J_k(n, r)$, which is defined by $\mathcal{L}_1, \ldots, \mathcal{L}_k$ and a natural number r that is less than or equal to n, we can obtain $\mathcal{O}\left(T_{\mathtt{S}} + (1/\sqrt{r/n})^k \cdot ((1/\sqrt{r}) \cdot T_{\mathtt{U}} + T_{\mathtt{C}})\right)$ in average time. Note that we assume $n \geq r$ following the previous study [16], and since k is a constant, we use the approximation $\delta \approx 1/r$.

The ISD algorithm is a random choice algorithm for the permutation matrix $\mathbf{P} \xleftarrow{\$} \mathbb{F}_2^{n \times n}$, and the algorithm must be run until a solution is found. We define the function $f : \mathbb{F}_2^{n \times n} \to \{0, 1\}$ of Grover's algorithm as $f(\mathbf{P}) = 1$ if $\mathbf{P}$ is appropriate and $f(\mathbf{P}) = 0$ otherwise. Then, we can reduce the time complexity of the ISD algorithm by using Grover's algorithm. In other words, Grover's algorithm enables us to achieve quadratic speed-up for finding a good permutation matrix $\mathbf{P}$. Moreover, we can use quantum walk to search the solution $\widetilde{\mathbf{e}}$ and construct Grover's algorithm.

Computational Cost. When we set T_f as the computational cost to operate a function f (the computational cost of each trial of the ISD algorithm) and set $\mathcal{P}$ as the probability that a permutation matrix $\mathbf{P} \leftarrow \mathbb{F}_2^{n \times n}$ is appropriate, the time complexity of the quantum ISD algorithm is expressed as $\mathrm{O}(T_f/\sqrt{\mathcal{P}})$. Bernstein [3] used Grover's algorithm and developed the quantum `Prange` with the time complexity $\widetilde{\mathcal{O}}(2^{0.060350n})$ and space complexity that is polynomial in n. Kachiga and Tillich [15] applied Grover's algorithm and the quantum walk to `Stern`, `MMT`, and `BJMM`. The quantum `Stern` (resp. `MMT`, `BJMM`) runs in time

$\widetilde{\mathcal{O}}(2^{0.059697n})$ (resp. $\widetilde{\mathcal{O}}(2^{0.059037n})$, $\widetilde{\mathcal{O}}(2^{0.058660n})$) with space $\widetilde{\mathcal{O}}(2^{0.0077375n})$ (resp. $\widetilde{\mathcal{O}}(2^{0.015574n})$, $\widetilde{\mathcal{O}}(2^{0.023413n})$).

4 Our Proposed Algorithm

In this section, we propose a quantum ISD algorithm that is a half quantumized variant of `BM18`. Our proposed algorithm utilizes Grover's algorithm and the quantum walk as the other quantum ISD algorithms [3,15]. We explain how to construct the function f. At first, we set seven parameters

$$\begin{aligned} p_1, p_2 &\in \{0, \ldots, w\}, & \ell_1, \ell_2 &\in \{0, \ldots, n-k\}, \\ w_1^{(1)}, w_1^{(2)} &\in \{0, \ldots, \ell_1\}, & w_2^{(2)} &\in \{0, \ldots, \ell_2\}, \end{aligned}$$

with the following four conditions:

(a) $p_1 \geq p_2/2$,
(b) $\ell_1 + \ell_2 = n - k$,
(c) $w_1^{(1)} \geq w_1^{(2)}/2$,
(d) $w_1^{(2)} + w_2^{(2)} = w - p_2$,

and the inequality

$$\binom{p_2}{p_2/2}\binom{k-p_2}{p_1-p_2/2} \geq 2^{\ell_1} \cdot \left(\binom{w_1^{(2)}}{w_1^{(2)}/2}\binom{\ell_1 - w_1^{(2)}}{w_1^{(1)} - w_1^{(1)}/2}\right)^{-1}$$

(used in [4, Lemma 5.1]). The function f consists of the following four steps:

(Step 2-1: Elementary Operation) We apply elementary row operations to a matrix $\mathbf{HP}$ and obtain $(\widetilde{\mathbf{Q}} \mid \mathbf{I}_{n-k})$, where $\widetilde{\mathbf{Q}} \in \mathbb{F}_2^{(n-k)\times k}$. If we can obtain a matrix of the desired form, there is a non-singular matrix $\mathbf{T} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ satisfying

$$\mathbf{THP} = (\widetilde{\mathbf{Q}} \mid \mathbf{I}_{n-k}).$$

We set

$$\tilde{\mathbf{s}} \coloneqq \mathbf{Ts}.$$

Step 2-2: Constructing Lists) We construct four lists

- $\mathcal{L}_1^{(0)} = \{(\tilde{\mathbf{e}}_1^{(0)}, \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_1^{(0)}) \in \mathbb{F}_2^{k/2} \times \mathbf{0}^{k/2} \times \mathbb{F}_2^{n-k} \mid \mathrm{wt}(\tilde{\mathbf{e}}_1^{(0)}) = p_1/2\}$,
- $\mathcal{L}_2^{(0)} = \{(\tilde{\mathbf{e}}_2^{(0)}, \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_2^{(0)}) \in \mathbf{0}^{k/2} \times \mathbb{F}_2^{k/2} \times \mathbb{F}_2^{n-k} \mid \mathrm{wt}(\tilde{\mathbf{e}}_2^{(0)}) = p_1/2\}$,
- $\mathcal{L}_3^{(0)} = \{(\tilde{\mathbf{e}}_3^{(0)}, \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_3^{(0)}) \in \mathbb{F}_2^{k/2} \times \mathbf{0}^{k/2} \times \mathbb{F}_2^{n-k} \mid \mathrm{wt}(\tilde{\mathbf{e}}_3^{(0)}) = p_1/2\}$,
- $\mathcal{L}_4^{(0)} = \{(\tilde{\mathbf{e}}_4^{(0)}, \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_4^{(0)} + \tilde{\mathbf{s}}) \in \mathbf{0}^{k/2} \times \mathbb{F}_2^{k/2} \times \mathbb{F}_2^{n-k} \mid \mathrm{wt}(\tilde{\mathbf{e}}_4^{(0)}) = p_1/2\}$.

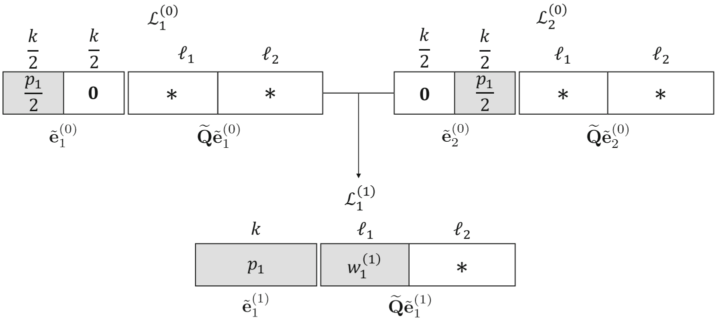


Fig. 2. Overview for constructing $\tilde{\mathbf{e}}_1^{(1)}$ from $\tilde{\mathbf{e}}_1^{(0)}$ and $\tilde{\mathbf{e}}_2^{(0)}$ in the Step 2–3

We further prepare two empty lists $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$. For all n-dimensional vectors in these lists, we divide them into three blocks as the first k, middle ℓ_1, and last $\ell_2 = n - k - \ell_1$ coordinates. For a list $\mathcal{L}_1^{(0)}$, we use $(\widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_1^{(0)})_{\ell_1}$ and $(\widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_1^{(0)})_{\ell_2}$ to denote the middle block and last block of list vectors, respectively. We also use the same notations for the other lists.

(Step 2–3: Merging Lists) For the tuples of lists $(\mathcal{L}_1^{(0)}, \mathcal{L}_2^{(0)})$ and $(\mathcal{L}_3^{(0)}, \mathcal{L}_4^{(0)})$, we apply the *classical* $(w_1^{(1)}/n, \ell_1)$-NN algorithm to the middle ℓ_1 coordinates and obtain two lists

- $\mathcal{L}_1^{(1)} = \left\{ (\tilde{\mathbf{e}}_1^{(1)}, \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_1^{(1)}) \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k} \;\middle|\; \begin{array}{c} \mathrm{wt}(\tilde{\mathbf{e}}_1^{(1)}) = p_1, \\ \mathrm{wt}((\widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_1^{(1)})_{\ell_1}) = w_1^{(1)} \end{array} \right\}$,
- $\mathcal{L}_2^{(1)} = \left\{ (\tilde{\mathbf{e}}_2^{(1)}, \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_2^{(1)} + \tilde{\mathbf{s}}) \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k} \;\middle|\; \begin{array}{c} \mathrm{wt}(\tilde{\mathbf{e}}_2^{(1)}) = p_1, \\ \mathrm{wt}((\widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_2^{(1)} + \tilde{\mathbf{s}})_{\ell_1}) = w_1^{(1)} \end{array} \right\}$.

Figure 2 illustrates the overview for constructing $\tilde{\mathbf{e}}_1^{(1)}$.

(Step 2–4: Search via Quantum Walk) For the tuple of lists $(\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)})$ and $(\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)})$, we apply the Kirshanova's quantum walk $(w_2^{(2)}/n, \ell_2)$-NN algorithm to the last ℓ_2 coordinates and obtain the list

- $\mathcal{L}^{(2)} = \left\{ (\tilde{\mathbf{e}}', \tilde{\mathbf{e}}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k} \;\middle|\; \tilde{\mathbf{e}}'' = \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}' + \tilde{\mathbf{s}}, \mathrm{wt}((\tilde{\mathbf{e}}'')_{\ell_2}) = w_2^{(2)} \right\}$.

We define $f(\mathbf{P}) = 1$ if there is a vector $(\tilde{\mathbf{e}}', \tilde{\mathbf{e}}'') \in \mathcal{L}^{(2)}$ such that $\mathrm{wt}(\tilde{\mathbf{e}}') = p_2$ and $\mathrm{wt}((\tilde{\mathbf{e}}'')_{\ell_1}) = w_1^{(2)}$, and $f(\mathbf{P}) = 0$ otherwise. Figure 3 illustrates the overview for constructing $\tilde{\mathbf{e}}'$.

Correctness. Since we defined $\tilde{\mathbf{e}}_1^{(0)}, \tilde{\mathbf{e}}_3^{(0)} \in \mathbb{F}_2^{k/2} \times \mathbf{0}^{k/2}$ and $\tilde{\mathbf{e}}_2^{(0)}, \tilde{\mathbf{e}}_4^{(0)} \in \mathbf{0}^{k/2} \times \mathbb{F}_2^{k/2}$ in Step 2-2, it holds that $\mathrm{wt}(\tilde{\mathbf{e}}_1^{(1)}) = \mathrm{wt}(\tilde{\mathbf{e}}_2^{(1)}) = p_1$ in Step 2–3. Due to the condition (d), it holds that $\mathrm{wt}(\tilde{\mathbf{e}}'') = \mathrm{wt}((\tilde{\mathbf{e}}'')_{\ell_1}) + \mathrm{wt}((\tilde{\mathbf{e}}'')_{\ell_2}) = w_1^{(2)} + w_2^{(2)} = w - p_2$ in Step 2–4. By following the same argument as in Sect. 2, $\mathbf{e} = \mathbf{P}(\tilde{\mathbf{e}}' \mid \tilde{\mathbf{e}}'')$ is the correct solution of the syndrome decoding problem.

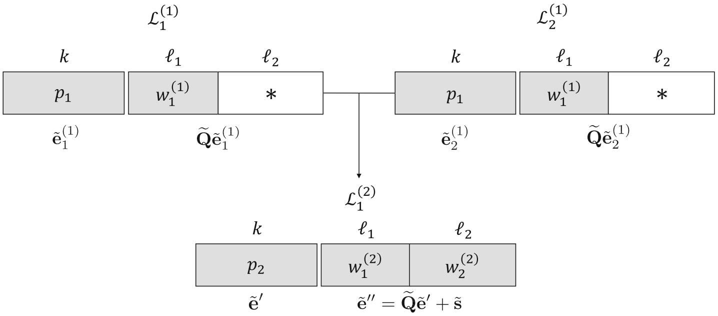

$\mathcal{L}_1^{(1)}$
$\mathcal{L}_2^{(1)}$
k
ℓ_1
ℓ_2
p_1
$w_1^{(1)}$
$*$
$\tilde{\mathbf{e}}_1^{(1)}$
$\widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_1^{(1)}$
$\tilde{\mathbf{e}}_2^{(1)}$
$\widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_2^{(1)}$
$\mathcal{L}_1^{(2)}$
p_2
$w_1^{(2)}$
$w_2^{(2)}$
$\tilde{\mathbf{e}}'$
$\tilde{\mathbf{e}}'' = \widetilde{\mathbf{Q}}\tilde{\mathbf{e}}' + \tilde{\mathbf{s}}$

Fig. 3. Overview for constructing $\tilde{\mathbf{e}}', \tilde{\mathbf{e}}''$ from $\tilde{\mathbf{e}}_1^{(1)}$ and $\tilde{\mathbf{e}}_2^{(1)}$ in the Step 2–4

Computational Cost. We evaluate the computational cost of the proposed ISD algorithm. The time complexity T_1 for evaluating the function f in Step 2-1 is polynomial in n, and the time complexity for the other steps become the following T_2, T_3, T_4:

Step 2-2: As the case of the *classical* `BM18` [4,6], it holds that $T_2 = \widetilde{\mathcal{O}}(S_0)$, where

$$S_0 \coloneqq \binom{k/2}{p_1/2}.$$

Step 2–3: As the case of the *classical* `BM18` [4,6], we assume that $\widetilde{\mathbf{Q}}\tilde{\mathbf{e}}_1^{(1)}$ and $\widetilde{\mathbf{Q}}\tilde{\mathbf{e}}^{(1)} + \mathbf{Ts}$ are distributed over $\mathbb{F}_2^{n-k}$ uniformly at random. Then, it holds that

$$T_3 = \widetilde{\mathcal{O}}\left(2^{\tau_{\mathrm{NN}}((\log_2 S_0)/n, \ell_1/n, w_1^{(1)}/n)\cdot n}\right).$$

Step 2–4: Based on Kirshanova's result [16], it holds that

$$T_4 = \widetilde{\mathcal{O}}(2^{(1-H(\mathbf{p}(\alpha,\beta,\gamma))+H(\mathbf{q}(\gamma)))\cdot n}),$$

where

$$S_1 \coloneqq \binom{k}{p_1}\binom{\ell_1}{w_1^{(1)}}2^{-\ell_1}, \quad r \coloneqq S_1^{\frac{2}{3}},$$

$$\alpha \coloneqq H^{-1}\left(1 - \log_2(r^{1/\ell_2})\right), \quad \beta \coloneqq H^{-1}\left(1 - \frac{3}{4}\log_2(r^{1/\ell_2})\right), \quad \gamma \coloneqq \frac{w_2^{(2)}}{\ell_2}.$$

Here, we use the notations defined in Sect. 3.

By definition, the time complexity for evaluating the function f is roughly

$$T_f \approx \max\{T_1, T_2, T_3, T_4\}.$$

As the case of the *classical* BM18 [4,6], the probability that a permutation matrix $\mathbf{P} \xleftarrow{\$} \mathbb{F}_2^{n\times n}$ becomes appropriate is

$$\mathcal{P} = \binom{k}{p_2}\binom{\ell_1}{w_1^{(2)}}\binom{\ell_2}{w_2^{(2)}}\binom{n}{w}^{-1}.$$

The total time complexity of the proposed algorithm is $\mathcal{O}(T_f/\sqrt{\mathcal{P}})$.

Next, we evaluate the space complexity of the proposed algorithm. In the Step 1 and Step 2-1, the space complexity is polynomial in n. As the case of the *classical* BM18 [4,6], the space complexities in the Steps 2 and 3 are $\widetilde{\mathcal{O}}(S_0)$ and $\widetilde{\mathcal{O}}(S_1)$, respectively. Based on Kirshanova's result [16], the space complexity in the Step 4 is $\widetilde{\mathcal{O}}(2^{(1-H(\mathbf{p}(\alpha,\beta,\gamma))+H(\mathbf{q}(\gamma)))\cdot n})$, where we use the values of α, β, γ as analyzed above.

As we discussed in Sect. 2, when we use the notations $\widetilde{\mathcal{O}}(2^{tn})$ and $\widetilde{\mathcal{O}}(2^{sn})$ to denote the time and space complexities to run the proposed algorithm, it holds that $t = p \cdot \max\{t_1, t_2, t_3\}, s = \max\{s_1, s_2\}$. As we claimed in Sect. 2, we set $w = n \cdot {H_2}^{-1}(1 - k/n)$ and use the approximation $\binom{n}{r} = \widetilde{\mathcal{O}}(2^{n\cdot H_2(r/n)})$ by following previous works. When we use the notation $c_x \coloneqq x/n$ for each parameter x, we have

$$\begin{aligned}
p &= c_k \cdot H_2\left(\frac{c_{p_2}}{c_k}\right) + c_{\ell_1} \cdot H_2\left(\frac{c_{w_1^{(2)}}}{c_{\ell_1}}\right) + c_{\ell_2} \cdot H_2\left(\frac{c_{w_2^{(2)}}}{c_{\ell_2}}\right) - H_2(c_w),\\
t_1 &= c_k \cdot H_2\left(\frac{c_{p_1}}{c_k}\right), \qquad t_2 = \tau_{\text{NN}}\left(c_k \cdot H_2\left(\frac{c_{p_1}}{c_k}\right), c_{\ell_1}, c_{w_1^{(1)}}\right),\\
t_3 &= 1 - H(\mathbf{p}(\alpha,\beta,\gamma)) + H(\mathbf{q}(\gamma))),\\
s_1 &= c_k \cdot H_2\left(\frac{c_{p_1}}{c_k}\right), \qquad s_2 = c_k \cdot H_2\left(\frac{c_{p_2}}{c_k}\right) + c_{\ell_1} \cdot H_2\left(\frac{c_{w^{(1)}1}}{c_{\ell_1}}\right) - \ell_1.
\end{aligned}$$

We numerically analyze the parameters to minimize the time complexity. When we set

$$\begin{aligned}
&c_k = 0.454, \quad c_w = 0.125869, \quad c_{p_1} = 0.000238090, \quad c_{p_2} = 0.000475004,\\
&c_{\ell_1} = 0.00150036, \quad c_{\ell_2} = 0.544500, \quad c_{w_1^{(1)}} = 0.00000552440,\\
&c_{w^{(1)}} = 0.0000107316, \quad c_{w^{(2)}} = 0.125384,
\end{aligned}$$

we have the time complexity $\widetilde{\mathcal{O}}(2^{0.059809n})$ and space complexity $\widetilde{\mathcal{O}}(2^{0.0014901n})$, while the classical BM18 has the time complexity $\widetilde{\mathcal{O}}(2^{0.0982n})$ and space complexity $\widetilde{\mathcal{O}}(2^{0.0717n})$.

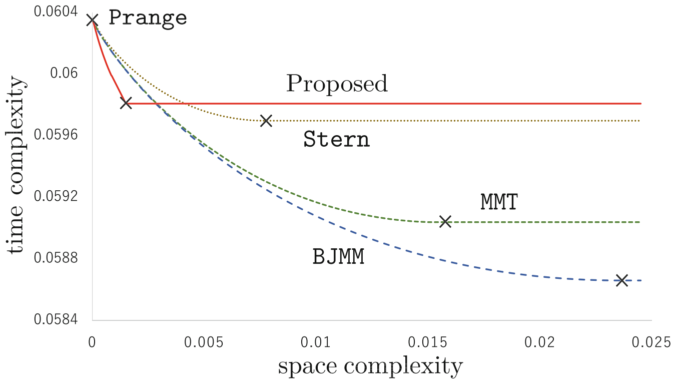


Fig. 4. Comparison of the time/space complexity of the quantum ISD algorithms

5 Comparison

As we discussed in Sect. 4, the proposed algorithm runs in time $\widetilde{\mathcal{O}}(2^{0.059809n})$ with the space $\widetilde{\mathcal{O}}(2^{0.0014901n})$. However, as we discussed in Sect. 3, the quantum `Stern`, `MMT`, and `BJMM` run in time $\widetilde{\mathcal{O}}(2^{0.059697n})$, $\widetilde{\mathcal{O}}(2^{0.059037n})$, and $\widetilde{\mathcal{O}}(2^{0.058660n})$ with the space $\widetilde{\mathcal{O}}(2^{0.0077375n})$, $\widetilde{\mathcal{O}}(2^{0.015574n})$, and $\widetilde{\mathcal{O}}(2^{0.023413n})$, respectively. Thus, the proposed algorithm is slower than the known ones when we focus on the minimum time complexity with sufficient space. Nevertheless, we believe that the proposed algorithm is valuable since it requires less space complexity. To justify the claim, we analyze parameters to minimize time complexity of each algorithm when the space complexity is limited. Figure 4 shows the comparison of the results.[7] Here, the × marks indicate the points, where the time complexity did not decrease even if we allow larger space complexity. As the figure indicates, the proposed algorithm is the fastest when the space complexity is limited. In particular, when the space complexity is limited by $\widetilde{\mathcal{O}}(2^{0.001490n})$, the proposed algorithm is the fastest ISD algorithm.

Acknowledgement. This work was partially supported by JSPS KAKENHI Grant Number 19K20267 and 21H03440, Japan, and JST CREST Grant Number JPMJCR2113, Japan.

References

1. Amico, M., Saleem, Z.H., Kumph, M.: Experimental study of Shor's factoring algorithm using the IBM Q experience. Phys. Rev. A **100**, 012305 (2019)

[7] Kirshanova proposed other quantum variants of `Stern` (Sect. 4 of [16]) that run in time $\widetilde{\mathcal{O}}(2^{0.059922n})$ (resp. $\widetilde{\mathcal{O}}(2^{0.059922n})$) with space $\widetilde{\mathcal{O}}(2^{0.00897n})$ (resp. $\widetilde{\mathcal{O}}(2^{0.00808n})$). Although we tried to find the optimized parameters to obtain the computational complexity via brute force search, we could not find them. Thus, we do not compare our result with these algorithms.

2. 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: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_31
3. Bernstein, D.J.: Grover vs. McEliece. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 73–80. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12929-2_6
4. Both, L.: Solvin k-list problems and their impact on information set decoding. Ph.D. thesis, Ruhr University Bochum, Germany (2018)
5. Both, L., May, A.: Optimizing BJMM with nearest neighbors : full decoding in $2^{2n/21}$ and MCEliece security. In: The Tenth International Workshop on Coding and Cryptography (WCC2017) (2017)
6. Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security. In: Lange, T., Steinwandt, R. (eds.) PQCrypto 2018. LNCS, vol. 10786, pp. 25–46. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79063-3_2
7. Carrier, K., Debris-Alazard, T., Meyer-Hilfiger, C., Tillich, J.: Statistical decoding 2.0: reducing decoding to LPN. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022–28th International Conference on the Theory and Application of Cryptology and Information Security. Lecture Notes in Computer Science, vol. 13794, pp. 477–507. Springer (2022). https://doi.org/10.1007/978-3-031-22972-5_17
8. Dumer, I.: On minimum distance decoding of linear codes. In: Proceedings of the 5th Joint Soviet-Swedish International Workshop Information Theory, pp. 50–52 (1991)
9. Esser, A.: Revisiting nearest-neighbor-based information set decoding. IACR Cryptol. ePrint Arch, 1328 (2022)
10. Esser, A., Bellini, E.: Syndrome decoding estimator. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography - PKC 2022–25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13177, pp. 112–141. Springer (2022). https://doi.org/10.1007/978-3-030-97121-2_5
11. Esser, A., May, A., Zweydinger, F.: Mceliece needs a break - solving mceliece-1284 and quasi-cyclic-2918 with modern ISD. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology - EUROCRYPT 2022–41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13277, pp. 433–457. Springer (2022). https://doi.org/10.1007/978-3-031-07082-2_16
12. Esser, A., Ramos-Calderer, S., Bellini, E., Latorre, J.I., Manzano, M.: Hybrid decoding - classical-quantum trade-offs for information set decoding. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Proceedings. Lecture Notes in Computer Science, vol. 13512, pp. 3–23. Springer (2022). https://doi.org/10.1007/978-3-031-17234-2_1
13. Finiasz, M., Sendrier, N.: Security bounds for the design of code-based cryptosystems. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 88–105. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_6
14. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing. pp. 212–219. ACM (1996)

15. Kachigar, G., Tillich, J.-P.: Quantum information set decoding algorithms. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 69–89. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_5
16. Kirshanova, E.: Improved quantum information set decoding. In: Lange, T., Steinwandt, R. (eds.) PQCrypto 2018. LNCS, vol. 10786, pp. 507–527. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79063-3_24
17. Lee, P.J., Brickell, E.F.: An observation on the security of McEliece's public-key cryptosystem. In: Barstow, D., Brauer, W., Brinch Hansen, P., Gries, D., Luckham, D., Moler, C., Pnueli, A., Seegmüller, G., Stoer, J., Wirth, N., Günther, C.G. (eds.) EUROCRYPT 1988. LNCS, vol. 330, pp. 275–280. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-45961-8_25
18. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. SIAM J. Comput. **40**(1), 142–164 (2011)
19. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_6
20. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 203–228. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_9
21. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. Deep Space Netw. Prog. Rep. **44**, 114–116 (1978)
22. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. Probl. Control Inf. Theory **15**(2), 159–166 (1986)
23. NIST. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals
24. NIST. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions
25. Prange, E.: The use of information sets in decoding cyclic codes. IRE Trans. Inf. Theory **8**(5), 5–9 (1962)
26. Schroeppel, R., Shamir, A.: A $T = O(2^{n/2}), s = O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM J. Comput. **10**(3), 456–464 (1981)
27. 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)
28. Stern, J.: A method for finding codewords of small weight. In: Cohen, G.D., Wolfmann, J. (eds.) Coding Theory and Applications, 3rd International Colloquium, Proceedings. Lecture Notes in Computer Science, vol. 388, pp. 106–113. Springer, Berlin (1988). https://doi.org/10.1007/BFb0019850

Cryptographic Protocols

CSI-SharK: CSI-FiSh with Sharing-friendly Keys

Shahla Atapoor[1], Karim Baghery[1](✉), Daniele Cozzo[1,2], and Robi Pedersen[1]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
{shahla.atapoor,karim.baghery,daniele.cozzo,robi.pedersen}@kuleuven.be
[2] IMDEA Software Institute, Madrid, Spain
daniele.cozzo@imdea.org

Abstract. CSI-FiSh is one of the most efficient isogeny-based signature schemes, which is proven to be secure in the Quantum Random Oracle Model (QROM). However, there is a bottleneck in CSI-FiSh in the threshold setting, which is that its public key needs to be generated by using $k-1$ secret keys. This leads to very inefficient threshold key generation protocols and also forces the parties to store $k-1$ secret shares. We present CSI-SharK, a new variant of **CSI**-FiSh that has more **Shar**ing-friendly **K**eys and is as efficient as the original scheme. This is accomplished by modifying the public key of the ID protocol, used in the original CSI-FiSh, to the equal length Structured Public Key (SPK), generated by a *single* secret key, and then proving that the modified ID protocol and the resulting signature scheme remain secure in the QROM. We translate existing CSI-FiSh-based threshold signatures and Distributed Key Generation (DKG) protocols to the CSI-SharK setting. We find that DKG schemes based on CSI-SharK outperform the state-of-the-art actively secure DKG protocols from the literature by a factor of about 3, while also strongly reducing the communication cost between the parties. We also uncover and discuss a flaw in the key generation of the actively secure CSI-FiSh based threshold signature *Sashimi*, that can prevent parties from signing. Finally, we discuss how (distributed) key generation and signature schemes in the isogeny setting are strongly parallelizable and we show that by using C independent CPU threads, the total runtime of such schemes can basically be reduced by a factor C. As multiple threads are standard in modern CPU architecture, this parallelizability is a strong incentive towards using isogeny-based (distributed) key generation and signature schemes in practical scenarios.

Keywords: Isogeny-based cryptography · DKG · Threshold Schemes · CSIDH

1 Introduction

Following Shor's attack [33] on the Factoring and Discrete Logarithm (DL) problems, researchers started exploring post-quantum cryptographic techniques to construct primitives and protocols that can be secure in the presence of

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 471–502, 2023.
https://doi.org/10.1007/978-3-031-35486-1_21

quantum adversaries. One of these areas is isogeny-based cryptography, which was initially proposed by Couveignes [11] and by Rostovtsev and Stolbunov [30, 36]. These proposals use hardness assumptions based on isogenies between ordinary elliptic curves, but were strongly weakened by the quantum attack of Childs, Jao and Soukarev [10] a few years later. Later proposals by Jao and De Feo [24] (SIDH) and Castryck et al. [8] (CSIDH) use supersingular elliptic curves instead of ordinary ones. While the attack by Childs et al. is still applicable to the latter, the authors of [8] managed to push the secure parameter set back into practical sizes. SIDH on the other hand was recently broken by a different attack, designed by Castryck and Decru [7] and further improved in subsequent works [25,29]. These attacks do not apply to CSIDH-based schemes.

Based on CSIDH, two signature schemes called SeaSign [15] and CSI-FiSh [4] have been proposed. The basic version of both these signature schemes are based on the same ID protocol that was introduced in [30] and both papers discuss extending the public keys in order to reduce the soundness error rate by using multiple secret keys. To allow this extension, the underlying language in the ID protocol is changed, and a witness for the Multi-Target Group Action Inverse Problem (MT-GAIP) [4,15] is proven. The fundamental difference between SeaSign and CSI-FiSh is that in the latter, the authors have performed a record computation of the underlying ideal-class group, which allows uniform sampling and canonical representations of ideals. As a result, unlike SeaSign, CSI-FiSh does not need to rely on rejection sampling and was therefore one of the first practical signature scheme based on isogenies, with signing times just below 3 s in the basic version of the scheme and a signature size of 1.8 KB. With an extended public key, signing can even be done in a few hundred milliseconds (Table 3 in [4]). Other practical isogeny-based signature schemes include [22,39] and later [16]. Recently, Baghery, Cozzo and Pedersen [2] presented a new extension of the basic ID protocol used in CSI-FiSh. They extend the public key to a *Structured Public Key* (SPK), which is generated using a single secret key and a set of public integers. This allows working with a larger challenge space and efficiently proving the actual witness, rather than the differences of two curves, as in MT-GAIP. However, to extract the single secret key, they require a Trusted Third Party (TTP) to generate the keys. Furthermore, to prove the security of their ID protocol, they rely on a new computational assumption, so-called $\mathbb{C}_k$-Vectorization Problem with Auxiliary Inputs ($\mathbb{C}_k$-VPwAI, see Definition 3.2), which its hardness recently was analysed in [19].

By virtue of its flexibility, CSI-FiSh is used as the base scheme to build various isogeny-based protocols, such as threshold signatures [12,17], lossy signatures [20], and forward secure ones [32]. Due to a wide range of applications (e.g. in blockchain), Threshold Signature Schemes (TSSs) have received more attention in recent years. Such schemes allow distributing the secret key into shares among multiple parties or devices, such that only a set of authorized parties can jointly sign a message to produce a single signature. Key recovery attacks on TSSs require more effort than the non-threshold ones, as the adversary has to attack more than one device or party simultaneously. A large amount of effort has been put on constructing threshold versions of classic schemes based on discrete logarithms and RSA. Most of these schemes could be integrated with a Distributed Key Generation (DKG) protocol, which was initially proposed by Pedersen in [27]. A secure version of it along with robustness guarantees was proposed by Gennaro et al. [23]. On the other hand, research on threshold isogeny-based signatures is very young. In 2020, De Feo and Meyer [17] proposed a threshold version of CSI-FiSh using Shamir secret sharing. The protocol is almost as efficient as a CSI-FiSh signature but only passively secure. Cozzo and Smart [12] then proposed Sashimi, the first actively secure threshold signature scheme with abort based on CSI-FiSh, that uses replicated secret sharing. Sashimi uses Zero-Knowledge (ZK) proofs to achieve active security, resulting in the rather elevated signing times of around 5 min for two parties. The cost of their DKG step is even higher, but was not estimated by the authors. Later, Beullens et al. proposed CSI-RAShi [3], an actively secure DKG protocol for CSI-FiSh using Shamir secret sharing. By introducing a new primitive called piecewise verifiable proofs, the authors manage to reduce the cost of generating public keys to about 18 s per party. Still, Sashimi [12] as well as CSI-RAShi [3], when used for extended public keys, remain quite inefficient to be used in practice.

1.1 Our Contributions and Techniques

CSI-SharK Signature Scheme. As our initial contribution, we present *CSI-SharK* in Sect. 3, a new variant of **CSI**-FiSh [4] that has more **Shar**ing-friendly **K**eys. To this end, we slightly modify the ID protocol used in CSI-FiSh [4] and then prove that the modified ID protocol and the resulting signature scheme remain secure in the QROM. Our main modification is to change the public key of CSI-FiSh to a SPK, $(E_0, E_1 = [c_1 \cdot \mathsf{sk}]E_0, \cdots, E_{k-1} = [c_{k-1} \cdot \mathsf{sk}]E_0)$, which is generated using a *single* secret key sk and $k-1$ public integers $c_1, \cdots, c_{k-1}$, instead of using $k-1$ distinct secret keys. Then we prove that the modified ID protocol is special sound for the language of MT-GAIP, and Honest-Verifier Zero-Knowledge (HVZK) for the language of $\mathbb{C}_k$-VPwAI. Roughly speaking, the modified ID protocol achieves to the best of the two ID protocols presented in [4] and [2]. In comparison with [4], the modified ID protocol has only a single secret key but needs to rely on an additional hardness assumption. In comparison with the one proposed in [2], it does not require a TTP, but only proves knowledge of a witness for the MT-GAIP, which is sufficient for building a signature scheme. We turn the modified ID protocol into a signature scheme in the standard manner using the Fiat–Shamir transform [21]. Our results and analysis show that, due to having a SPK and a single secret key, CSI-SharK is a suitable replacement for CSI-FiSh and achieves the same efficiency as the original scheme, without the need of storing multiple secret keys. The real advantage of CSI-SharK however manifests itself when it is used to build threshold schemes, which can be much more efficient than in the CSI-FiSh setting.

Threshold Schemes. As a next contribution, we show how to distribute key generation schemes and signatures using structured public keys. To this end, we first review the state-of-the-art DKG schemes, i.e. the passively secure scheme by

De Feo and Meyer [17], as well as the actively secure Sashimi [12] and CSI-RAShi [3]. We notice that there is a flaw in the current description of the DKG protocol in Sashimi that can lead to an attack, which could allow an adversary to prevent an honest party from using its share to sign. We then describe a version of Sashimi that we use as an actively secure full-threshold signature scheme with abort, preventing the attack. Finally, we discuss optimizations of these state-of-the-art schemes to act as a baseline for comparison with our proposed schemes. Then, in Sect. 5, we show how using structured public keys decreases the overall computational and communication cost of DKG protocols. We then describe ways to further reduce the computational cost of these protocols. The final protocols have a computational cost decreased by about a factor 3, when compared to the most optimal implementations without a SPK, while furthermore reducing the communication cost between the different parties by a significant factor, which depends on the public key size. In the full version [1], we also discuss how to translate the signatures from [3,12,17] to our setting. We note that the computational cost of the signing algorithm is not decreased by using SPKs, but generally threshold protocols based on CSI-SharK (e.g. threshold signatures) are more efficient and we also discuss some optimizations that are applicable to both settings.

Parallel Computations in Isogeny-Based Schemes. Due to the round-robin computational structure in isogeny-based threshold schemes, the total latency of distributed protocols can be huge. Because of this issue, based on the estimations done in Sashimi [12], signing could take about 5 min with two parties. As the final contribution, presented in Sect. 6, we observe that many of the computations in

Table 1. Comparison of our proposed CSI-SharK-based key generation schemes and the existing ones based on CSI-FiSh, for n parties, a public key with $k-1$ elliptic curves and a security parameter λ. Next to displaying the number of necessary secret sharings (Sec. Shar.), we present the leading term in the computational cost of the public key generation (PKGen) in terms of group actions as well as the leading term in communication cost per party (Comm.) in bits (we assume $k \gg n$). The latter two mainly depend on the parameters k, n and λ, as well as $\log p$, the size of the underlying prime field $\mathbb{F}_p$. We further display the hardness assumptions underlying the security of the different schemes. These assumptions are formally introduced in Sect. 2 and Sect. 3. TTP: Trusted Third Party, DKG: Distributed Key Generation, SSS: Shamir Secret Sharing, FT: Full-Threshold secret sharing.

	(Distributed) Key Generation Schemes from Isogenies					
	[17]	Section 5.1	Sashimi [12]	Section 5.2	CSI-RAShi [3]	Section 5.3
Adversary:	Passive	Passive	Active	Active	Active	Active
Setup by:	TTP	TTP	DKG	DKG	DKG	DKG
Sec. Shar.:	$k-1$ SSS	1 SSS	$k-1$ FT	1 FT	$k-1$ SSS	1 SSS
PKGen:	$k-1$	$k-1$	$5kn\lambda$	$n(k+2\sqrt{k})\lambda$	$4kn\lambda$	$n(k+2\sqrt{k})\lambda$
Comm.:	$\frac{1}{2}k\log p$	$\frac{1}{2}\log p$	$k\lambda\log p$	$(k+\frac{1}{2}\sqrt{k}\lambda)\log p$	$k\lambda(8n+\log p)$	$(k+\frac{1}{2}\sqrt{k}\lambda)\log p$
Hardness Assumpt.:	MT-GAIP, P-DDHA	MT-GAIP, P-DDHA, $\mathbb{C}_k$-VPwAI	MT-GAIP, dGAIP	MT-GAIP, dGAIP, $\mathbb{C}_k$-VPwAI	MT-GAIP, dGAIP	MT-GAIP, $\mathbb{C}_k$-dGAIP, $\mathbb{C}_k$-VPwAI

both non-threshold and threshold signatures built from isogenies are parallelizable. In light of this observation, we propose a general parallel execution strategy to be used in the cases that *one* (as in CSI-FiSh or CSI-SharK) or *multiple* parties (as in TSSs or DKG protocols) need to compute multiple independent elliptic curves, e.g. $[x_1]E_0, \dots, [x_r]E_0$. For the single party case, the computation simply uses the different cores (or threads) of a multi-core CPU to compute a subset of curves, while in the multi-party case, different parties further can run their computations in parallel. Using the proposed strategy and several optimizations, we could significantly reduce the runtimes of CSI-SharK, CSI-FiSh, and their threshold variants, making them quite practical. We give an overview of the overall complexities in Table 1.

2 Preliminaries

In this section, we briefly review secret sharing schemes, commitment schemes and isogeny-based cryptography. For an introduction to sigma protocol and (threshold) signatures, we refer the reader to the full version of paper [1].

Secret Sharing Schemes. We define an (n, t)-threshold access structure as a set consisting of n players, where each player holds shares of a secret, so that any subset of t or more players can reconstruct this secret. We assume that all players are Probabilistic Polynomial Time (PPT) Turing machines. We revisit two well known ways for realizing a (n, t)-threshold access structure that we will use in our distributed signature schemes, namely the Shamir Secret Sharing (SSS) scheme [31] and the Full-Threshold Secret Sharing (Full-TSS) scheme. Unlike the traditional versions that are defined over a prime field, we will use them over an integer ring $\mathbb{Z}_N$.

Shamir Secret Sharing. In (n, t)-Shamir secret sharing, a common polynomial $f(x) \in \mathbb{Z}_N[x]$ of degree $t-1$ is chosen, such that the secret s is set to be its constant term, i.e. $s = f(0)$. Each party P_i for $i \in \{1, \cdots, n\}$ is assigned the secret share $s_i = f(i)$ Then any subset $S \subset \{1, \dots, n\}$ of at least t parties can reconstruct the secret s via Lagrange interpolation by computing $s = f(0) = \sum_{i \in S} s_i \cdot L_{0,i}^S$, where

$$L_{0,i}^S := \prod_{j \in S \setminus \{i\}} \frac{j}{j-i} \pmod N,$$

are Lagrange basis polynomials evaluated at 0. Any subset of less than t parties are not able to find $s = f(0)$, as this is information-theoretically hidden, even given $t-1$ shares. Since we will be working over the ring $\mathbb{Z}_N$ with N composite, the difference $j - i$ of any two elements in $i, j \in S$ must be invertible modulo N. If q is the smallest prime factor of N, it is enough to require that $n < q$.

Full-Threshold Secret Sharing. In a Full-TSS, a secret $s \in \mathbb{Z}_N$ is additively shared among $P_1, \dots, P_n$. Specifically, each party P_i holds a random share $s_i \in \mathbb{Z}_N$ such that $s = s_1 + \cdots + s_n \bmod N$. Clearly, all parties are required to recover the secret s and thus a Full-TSS realizes a (n, n)-threshold access structure. Since the shares are random, up to $n-1$ parties cannot get any information about the secret s, as the remaining share information-theoretically hides it.

Init: Given (Init, P_i, B) from all parties, this initializes a commitment functionality from party P_i to the parties in B. This is shown with $\mathcal{F}_{\mathsf{Commit}}^{i,B}$, if B is a singleton set $B = \{j\}$ then we write $\mathcal{F}_{\mathsf{Commit}}^{i,j}$, and if $B = \mathcal{P} \setminus \{i\}$ then we write $\mathcal{F}_{\mathsf{Commit}}^{P_i}$.

Commit: On input of $(\mathsf{Commit}, \mathsf{id}, \mathsf{data})$ from parties P_i and $(\mathsf{Commit}, \mathsf{id}, \perp)$ from all parties in B the functionality stores $(\mathsf{id}, \perp)$.

Open: On input of $(\mathsf{Commit}, \mathsf{id})$ from all players in $B \cup \{i\}$ the functionality retrieves the entry $(\mathsf{id}, \mathsf{data})$ and returns data to all parties in B.

Fig. 1. The Functionality $\mathcal{F}_{\mathsf{Commit}}$

Commitment Schemes. In our protocols, we assume parties have access to a commitment functionality $\mathcal{F}_{\mathsf{Commit}}$, which allows one party to commit, and later open the value to a set of parties. We assume the opened value is only available to the targeted receivers and is sent over a secure communication channel. The description of $\mathcal{F}_{\mathsf{Commit}}$ is provided in Fig. 1, which can be easily implemented in the random oracle model. This gives the commitment properties such as extractability and equivocability that are crucial for our security proofs.

Isogeny-Based Cryptography. Isogenies are rational maps between two elliptic curves $\varphi : E \rightarrow E'$, that are also homomorphisms with respect to the natural group structure of E and E' [14,35]. For simplicity, we introduce isogenies as an abstract cryptographic concept in this section and focus only on the high-level ideas that allow us to build cryptographic protocols on top of them. For a more thorough introduction, we refer the reader to [8] and [14].

Isogeny computations can be abstracted as a group $\mathcal{G}$ acting freely and transitively on a set $\mathcal{E}$ [11]. In the case relevant to this work, $\mathcal{E}$ denotes the set of supersingular elliptic curves over a finite field $\mathbb{F}_p$ where p is a large prime. $\mathcal{G}$ denotes the ideal-class group $\mathrm{Cl}(\mathcal{O})$, where $\mathcal{O}$ is an order of the quadratic imaginary field $\mathbb{Q}(\sqrt{-p})$, isomorphic to the $\mathbb{F}_p$-rational endomorphism ring $\mathsf{End}_{\mathbb{F}_p}(E)$ elliptic curves $E \in \mathcal{E}$. Isogenies are uniquely defined through the kernels of the ideals in $\mathcal{O}$. Throughout this work, we assume that the class group, as well as a generating ideal $\mathfrak{g}$ of it are known. In that case, we can define this group action as $[\cdot] : \mathbb{Z}_N \times \mathcal{E} \rightarrow \mathcal{E}$, where N is the size of the class group; asymptotically $N \approx \sqrt{p}$ [8,34]. As an example for the group action, we could identify the isogeny $\varphi : E \rightarrow E'$ with some element $a \in \mathbb{Z}_N$, such that $E' = [a]E$ holds. This means that the separable part of the isogeny φ has the kernel $\bigcap_{\alpha \in \mathfrak{g}^a} \ker \alpha$. Note that $[a]([b]E) = [a+b]E$. In cryptographic settings, this group action comes equipped with some hardness assumptions, that are also difficult for quantum computers.

Definition 2.1 (Group action inverse problem (GAIP) [8,15]). *Given two supersingular elliptic curves $E, E' \in \mathcal{E}$ over the same finite field $\mathbb{F}_p$ and with $\mathsf{End}_{\mathbb{F}_p}(E) \simeq \mathsf{End}_{\mathbb{F}_p}(E') \simeq \mathcal{O}$, find $a \in \mathbb{Z}_N$, such that $E' = [a]E$.*

Definition 2.2 (Decisional GAIP [3,37]). *Given a triple $([a]E, [b]E, [c]E)$, where $E \in \mathcal{E}$, with the same $\mathbb{F}_p$-rational endomorphism ring, decide whether $[c]E = [a+b]E$ or not.*

```
KeyGen($1^{\mathsf{sec}}$): Given $E_0$, the secret and public key are generated as follows:
    1. For $i = 1, \ldots, k-1$, sample $a_i \leftarrow \mathbb{Z}_N$ and compute $E_i = [a_i]E_0$.
    2. Return $\mathsf{sk} = (a_1 \ldots, a_{k-1})$, $\mathsf{pk} = (E_0, \ldots, E_{k-1})$.
Sign($\mathsf{sk}, m$): To sign a message $m$, the signer performs
    1. For $i = 1, \ldots, t_S$: sample $b_i \leftarrow \mathbb{Z}_N$, and set $E_{b_i} = [b_i]E_0$.
    2. Set $(d_1, \ldots, d_{t_S}) = \mathsf{H}(E_{b_1}, \ldots, E_{b_{t_S}} \parallel m)$.
    3. For $i = 1, \ldots, t_S$: set $r_i = b_i - \mathrm{sign}(d_i) a_{|d_i|} \pmod{N}$.
    4. Return $\{(r_i, d_i)\}_{i=1}^{t_S}$.
Verify($\{(r_i, d_i)\}_{i=1}^{t_S}, m, \mathsf{pk}$): To verify a signature $\{(r_i, d_i)\}_{i=1}^{t_S}$ on $m$, one performs:
    1. For $i = 1, \ldots, t_S$: compute $E'_{b_i} = [r_i]E_{d_i}$.
    2. $(d'_1, \ldots, d'_{t_S}) = \mathsf{H}(E'_{b_1}, \ldots, E'_{b_{t_S}} \| m)$.
    3. If $(d_1, \ldots, d_{t_S}) = (d'_1, \ldots, d'_{t_S})$ then return valid, else output invalid.
```

Fig. 2. CSI-FiSh Signature Scheme [4].

Isogenies are efficiently computable, but hard to invert. As such, they are a powerful tool to build post-quantum cryptographic protocols. One of these schemes is the isogeny-based signature scheme CSI-FiSh proposed by Beullens, Kleinjung and Vercauteren [4]. The basic version of CSI-FiSh is based on an ID protocol with binary challenges initially proposed in [37]. CSI-FiSh starts with the special elliptic curve $E_0 : y^2 = x^3 + x$. Public keys are created with the action of an element $a \in \mathbb{Z}_N$ on E_0. The owner of the public key $E = [a]E_0$ proves knowledge of the secret key a by a standard binary sigma protocol. This is then turned into a signature scheme using the Fiat–Shamir heuristic [21]. Note that the class group enjoys a symmetry around the elliptic curve E_0, as the elliptic curve $[-a]E_0$ is $\mathbb{F}_p$-isomorphic to the quadratic twist of $[a]E_0$, a map that is easily computable without any extra information. As a result, we can implicitly include the twist in the public key and extend the challenge space to $\{-1, 0, 1\}$, so that the soundness error rate is reduced to $\frac{1}{3}$.

To further reduce the number of repetitions needed to achieve a security level of 2^{sec}, the authors of [4] used a technique proposed in [15] and enlarged the public key by using multiple secret keys $(a_1, \ldots, a_{k-1})$ and an extended public-key of the $k-1$ associated elements. The resulting number of repetitions is $t_S = \lceil \mathsf{sec} \log_{2k-1} 2 \rceil$, although one can reduce t_S a little bit by choosing a 'slow' hash function. We present the CSI-FiSh in Fig. 2. Note that $\mathsf{H} : \{0,1\}^\star \to \{-(k-1), \ldots, k-1\}^{t_S \lceil \log(2k-1) \rceil}$ represents a hash function modeled as a random oracle. We simply denote the twist of a curve E_a as E_{-a}. The extension of the public key in this way leads to a change in the underlying language, and the prover now must prove that it knows a secret $s \in \mathbb{Z}_N$ such that $E_j = [s]E_i$ for some pair of elliptic curves appearing in the public key list. As a result, CSI-FiSh relies on the hardness of a multi-target version of GAIP, called MT-GAIP.

Definition 2.3 (MT-GAIP [4,15]). *Given k supersingular elliptic curves $E_0, E_1, \ldots, E_k \in \mathcal{E}$ over $\mathbb{F}_p$ with the same $\mathbb{F}_p$-rational endomorphism ring, find $a \in \mathbb{Z}_N$, such that $E_i = [a]E_j$ for some $i, j \in \{0, \ldots, k\}$ with $i \neq j$.*

CSI-FiSh is sEUF-CMA secure in the QROM under the MT-GAIP assumption [4]. For later reference, we also introduce the Power-Decision Diffie-Hellman (Power-DDHA) assumption defined in [17], which is used in some of our schemes.

Definition 2.4 (Power-DDHA). *Given an element $E \in \mathcal{E}$ and $a \in \mathbb{Z}_N$. The a-Power-DDHA problem is: given $(a, E, [s]E, F)$, where s is uniformly sampled from $\mathbb{Z}_N$ and where $F \in \mathcal{E}$, either sampled from the uniform distribution on $\mathcal{E}$ or $F = [as]E$, decide which distribution F is drawn from.*

For details on the quantum security of CSIDH-based schemes, see [6,8,9,28].

3 CSI-FiSh with Structured Public Keys

In this section, we revisit a different way of extending the public key (of the ID protocol) used in CSI-FiSh, where instead of sampling a total of $k-1$ different secrets $a_1, \ldots, a_{k-1}$, we use $k-1$ different multiples of the same secret key a to generate the public key. The idea to build such an ID protocol for proving the knowledge of a was first proposed by Baghery et al. [2], but needed a TTP to generate the public key in order to guarantee the correct structure. Such a *structured* public key (SPK) consists of an integer set $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \ldots, c_{k-1}\}$ and a set of elliptic curves[1] $\{E_i = [c_i a]E_0\}_{i=0,\ldots,k-1}$. In their ID protocol, to ensure extractability of the witness modulo a composite number N, the integer set $\mathbb{C}_{k-1}$ has to be an exceptional set [5,13], as defined below. When E_0 is the same special base curve as in CSI-FiSh, the authors further introduce the notation of superexceptional sets [2]. The latter case is referred to as a *symmetric* hard homogeneous space.

Definition 3.1 ((Super-)Exceptional set). *An* exceptional set modulo N *is a set $\mathbb{C}_{k-1} = \{c_0, \ldots, c_{k-1}\} \subseteq \mathbb{Z}_N$, where the pairwise difference $c_i - c_j$ of all elements $c_i \neq c_j$ is invertible modulo N. A* superexceptional set modulo N *is an exceptional set $\mathbb{C}_{k-1} = \{c_0, \ldots, c_{k-1}\}$, where also the pairwise sum $c_i + c_j$ of all elements c_i, c_j (including $c_i = c_j$) is invertible modulo N.*

The hardness of obtaining the secret key from a structured public key relies on the following computational problem which is defined in [2].

Definition 3.2 ($(c_0, c_1, \cdots, c_{k-1})$-Vectorization Problem with Auxiliary Inputs ($\mathbb{C}_{k-1}$-VPwAI)). *Given an element $E \in \mathcal{E}$ and the pairs $(c_i, [c_i x]E)_{i=1}^{k-1}$, where $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \ldots, c_{k-1}\}$ is an exceptional set, find $x \in \mathbb{Z}_N$.*

The ID protocol of Baghery et al. [2] is designed to allow the prover to efficiently prove the knowledge of the secret key a, while relying on the fact that the public key indeed has the desired structure. The authors of [2] solve this problem by letting either a TTP generate these keys, or alternatively by relying on computationally rather heavy *well-formedness proofs* for proving the correctness of the key generation. We refer to the original source for more details.

[1] For simplicity, we include E_0 in the public-key. Note that [0] simply denotes the neutral element of the group action.

Setup: Given E_0, sample a public (super)exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \cdots, c_{k-1}\}$; Sample $a \leftarrow \mathbb{Z}_N$ and for $i = 1, \ldots, k-1$ set $E_i = [c_i a]E_0$; Output $(\mathbb{C}_{k-1}, E_0, E_1, \ldots, E_{k-1})$.
Prover: Given $(a, (\mathbb{C}_{k-1}, E_0, \cdots, E_{k-1}))$ the prover samples $b \leftarrow \mathbb{Z}_N$, and sends $E_b = [b]E_0$ to the verifier.
Verifier: Given $(\mathbb{C}_{k-1}, E_0, \cdots, E_{k-1})$ and E_b the verifier samples a random challenge $d \leftarrow \{0, \ldots, k-1\}$ and sends it to the prover.
Prover: Given $(a, \mathbb{C}_{k-1}, d)$ the prover computes $r = b - c_d \cdot a \mod N$ and sends it to the verifier.
Verifier: Given $((\mathbb{C}_{k-1}, E_0, \cdots, E_{k-1}), E_b)$ and r, return $E_b \overset{?}{=} [r]E_d$.

Fig. 3. The Modified Identification Protocol with Structured Public Keys.

3.1 The Modified Identification Protocol

In this section, we construct an ID protocol that achieves the best of the two ID protocols constructed in [2] and [4]. The protocol of [2] uses structured public keys and soundness relies on the $\mathbb{C}_{k-1}$-VPwAI (Definition 3.2), while the protocol from [4] uses non-structured (extended) public keys and soundness relies on MT-GAIP. Note that in order to guarantee that the public keys in the former actually have the correct structure, the scheme either relies on trusted third parties generating the keys or on heavy proofs of *well-formedness.*

The idea behind our new ID protocol is to work with structured public keys, but nevertheless base the soundness of the protocol on MT-GAIP. The advantage of relying on MT-GAIP, rather than on $\mathbb{C}_{k-1}$-VPwAI for soundness, is that we do not need a TTP, or heavy proofs of well-formedness, to generate the public keys. The SPK thus becomes a perk for the prover rather than a requirement for the protocol. The idea is simple, yet powerful: the prover first samples a secret key $a \leftarrow \mathbb{Z}_N$ and a public *(super-)exceptional* set $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \cdots, c_{k-1}\}$, then generates and publishes the SPK $(E_0, E_1, \ldots, E_{k-1})$, where $E_i = [c_i a]E_0$ for $i = 1 \ldots, k-1$. Next, instead of proving knowledge of a witness $a \in \mathbb{Z}_N$ for $\mathbb{C}_{k-1}$-VPwAI, i.e. the secret key underlying the SPK, as done in [2] and requiring the public key to be well-formed, the prover proves a witness for MT-GAIP, i.e. that it knows a secret $s \in \mathbb{Z}_N$ such that $E_j = [s]E_i$ for some pair of elliptic curves appearing in the public key, as in [4]. This prevents the prover from cheating, even if the public key would not actually be correctly structured. In the case where the public key would indeed be correctly structured, knowing a witness to the MT-GAIP instance also allows to extract a witness for $\mathbb{C}_{k-1}$-VPwAI. With this, we have circumvented the need to prove well-formedness of the SPK, a main shortcoming of the protocol in [2]. In comparison to the protocol in [4], our ID protocol has the advantage, that it only needs a single public key to be stored, independent of the size of the public key.

Interestingly, our new ID protocol is HVZK assuming that $\mathbb{C}_{k-1}$-VPwAI is hard. Figure 3 summarizes the resulting ID protocol.

Note that similar to the ID protocol used in CSI-FiSh, the key generation can be done by the prover. Therefore a malicious prover might choose to publish a public key that is not structured. However, based on MT-GAIP, without knowing s, it would be *still hard for an adversarial prover* to prove that it knows $s \in \mathbb{Z}_N$ such that $E_j = [s]E_i$ for some pair of elliptic curves in the public key. On the other hand, relying on the $\mathbb{C}_{k-1}$-VPwAI from Definition 3.2, we know that obtaining a from an honestly generated SPK is computationally hard. As a result, there is no real reason for the prover to generate its public key maliciously. In fact, due to several efficiency reasons that we will discuss in later sections, the prover is rather incentivized to sample the public key honestly.

Theorem 3.1. *The ID protocol presented in Fig. 3 is complete, special sound with soundness error rate $\frac{1}{k}$ for the language of MT-GAIP (Definition 2.3), and HVZK for the language of $\mathbb{C}_{k-1}$-VPwAI (Definition 3.2).*

Proof. The proof is analogous to the security proofs of the ID protocols discussed in [4] and [2]. However, similar to the ID protocol proposed in [2], we additionally rely on the hardness of $\mathbb{C}_{k-1}$-VPwAI [2], but without the need for a TTP.

Completeness. The honest prover knows the secret a for the public key $\{E_i = [c_i a]E_0\}_{i=0,\dots,k-1}$, where $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \cdots, c_{k-1}\}$ is a public *(super-)*

exceptional set. The honest verifier checks if $E_b \overset{?}{=} [r]E_d$. For an honestly generated proof, it holds that $[r]E_d = [b - c_d a]E_d = [b - c_d a][c_d a]E_0 = [b]E_0 = E_b$.

Honest-Verifier Zero-Knowledge. We construct a simulator that acts as follows: given the honestly sampled d, it samples r randomly from $\mathbb{Z}_N$; then sets $E_b = [r]E_d$ and returns the transcript (E_b, d, r). In both the real and the simulated transcripts, r and E_b are sampled uniformly at random, yielding indistinguishable distributions. Note that relying on the $\mathbb{C}_{k-1}$-VPwAI, obtaining the secret key a from an honestly generated SPK is computationally hard.

Special Soundness. Given two valid transcripts of the Σ-protocol, an efficient extraction algorithm extracts a witness as follows: Let (E_b, d, r) and (E_b, d', r') be two acceptable transcripts of the protocol, where $d \neq d'$, consequently $r \neq r'$ (for non-zero a). From the verification equation, one can conclude that $[r]E_d = [r']E_{d'}$ and consequently $E_d = [r' - r]E_{d'}$, which allows the extractor to obtain $r' - r$ as a solution to the MT-GAIP.

□

Remark 3.1. Note that the ID protocol used in CSI-Fish [4] is special sound and HVZK for the language of MT-GAIP, and under a trusted key generation the ID protocol proposed in [2] is special sound and HVZK for the language of $\mathbb{C}_{k-1}$-VPwAI. The issue with the former is that it requires $k - 1$ independent secret keys, and the concern with the later is that it needs a TTP to generate the keys. Our ID protocol achieves to the best of both, as it has a *single* secret key, and *does not* require a TTP, while relying on both assumptions.

```
$\mathsf{KeyGen}(1^n)$: Given $E_0$, the secret key and public are generated as follows:
    1. Sample $a \leftarrow \mathbb{Z}_N$;
    2. Generate a public (super)exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \cdots, c_{k-1}\}$;
    3. Given $E_0$, for each $c_i \in \mathbb{C}_{k-1}$ set $E_i = [c_i a]E_0$.
    4. Return $\mathsf{sk} = a$, $\mathsf{pk} = (E_0, E_1, \ldots, E_{k-1})$.
$\mathsf{Sign}(\mathsf{sk}, m)$: To sign a message $m$, the signer performs
    1. For $i = 1, 2, \ldots, t_S$: sample $b_i \leftarrow \mathbb{Z}_N$, and set $E_{b_i} = [b_i]E_0$.
    2. Set $(d_1, \ldots, d_{t_S}) = \mathsf{H}(E_{b_1}, \ldots, E_{b_{t_S}} \parallel m)$.
    3. For $i = 1, 2, \ldots, t_S$: set $r_i = b_i - c_{d_i} \cdot a \pmod{N}$.
    4. Return $\{(r_i, d_i)\}_{i=1}^{t_S}$.
$\mathsf{Verify}(\{(r_i, d_i)\}_{i=1}^{t}, m, \mathsf{pk})$: To verify a signature $\{(r_i, d_i)\}_{i=1}^{t_S}$ on $m$, one performs:
    1. For $i = 1, 2, \ldots, t_S$: compute $E'_{b_i} = [r_i]E_{d_i}$.
    2. $(d'_1, d'_2, \ldots, d'_{t_S}) = \mathsf{H}(E'_{b_1}, \ldots, E'_{b_{t_S}} \| m)$.
    3. If $(d_1, d_2, \ldots, d_{t_S}) = (d'_1, d'_2, \cdots, d'_{t_S})$ then return valid, else output invalid.
```

Fig. 4. CSI-SharK Signature Scheme.

3.2 NIZK Argument and Signature Scheme

Our ID protocol from Fig. 3 can be transformed into a non-interactive zero-knowledge argument or a signature scheme in the standard ways using the Fiat–Shamir transform [21]. To this end, we introduce the hash function $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^{t_S \lceil \log_2 k \rceil}$, modelled as a random oracle, where t_S denotes the number of repetitions needed in the protocol. For simplicity, we will only present the resulting signature. The NIZK argument can be constructed with a similar technique, and its security can be proven in a similar way as in Lemma 3.1 in [2]. We present our signature scheme in Fig. 4, and call it CSI-SharK, which stands for ***CSI**-FiSh with **Shar**ing-friendly **K**eys.*

Theorem 3.2. *Under MT-GAIP (Definition 2.3) and $\mathbb{C}_{k-1}$-VPwAI (Definition 3.2), when H is modelled as a (quantum) random oracle, then the CSI-SharK signature scheme described in Fig. 4 is sEUF-CMA secure.*

Proof. From the security of the resulting NIZK argument (similar to Lemma 3.2 in [2]), we know that the modified ID protocol has special soundness and unique responses. Then, by Theorem 25 in [18], it is a quantum argument of knowledge. Moreover, since the modified protocol has λ bits of min-entropy, from Theorem 22 of [18], this shows that the CSI-SharK obtained via Fiat–Shamir is sEUF-CMA in the QROM. □

Optimizations and Efficiency. Our basis protocol has a soundness error rate $\frac{1}{k}$. By choosing the unique base curve E_0 we can again increase this to a soundness error rate of $\frac{1}{2k-1}$ by also using the twists. This implies that we need $t_S = \lceil \mathsf{sec} \log_{2k-1} 2 \rceil$ protocol repetitions to reach a desired target soundness error of $2^{-\mathsf{sec}}$. To guarantee that the protocol is still HVZK and the SPK does not reveal any information about the secret key, we need to restrict $\mathbb{C}_{k-1}$ to superexceptional sets. As a result, the runtimes of our protocols are exactly the same as the respective versions from [4] or [2]. The same holds for the public-key and proof or signature sizes.

3.3 Proof of Commitments and Well-Formedness of SPK

In our actively secure distributed protocols, to guarantee that the parties follow the protocol, they are asked to commit to their secret shares and prove knowledge of the committed value. Furthermore, the parties will be required to prove that they indeed act with their committed secret value on some given elliptic curve to prove the correctness of generating/updating the SPK. More precisely, each party will need to prove knowledge of a witness s to the following language, which we define for arbitrary j and a public (super)exceptional set $\mathbb{C}_j$ with integer elements $\{c_1 = 1, c_2, \cdots, c_j\}$.

$$\mathbb{L}_j := \left\{ \begin{array}{c} \Big((F_0, F_0', E_1, E_1', \ldots, E_j, E_j', \mathbb{C}_j := \{c_1 = 1, c_2, \cdots, c_j\}),\ s\Big) : \\ (F_0' = [s]F_0) \bigwedge \Big(\bigwedge_{i=1}^{j} (E_i' = [c_i s]E_i)\Big) \end{array} \right\}. \tag{1}$$

In other words, since we set $c_1 = 1$, then the prover would need to prove in a zero-knowledge manner that it knows a unique witness s for

- an instance of the GAIP, when $j = 0$.
- two simultaneous instances of the GAIP, when $j = 1$.
- an instance for conjunction of the GAIP and $\mathbb{C}_{k-1}$-VPwAI, when $j = k - 1$.

The first item can be proven using the basic ID protocol from CSI-FiSh [4], while for the next two items we use the techniques of conjunctive Σ-protocols. We

```
ZK.$P_1((F_0, F_0', E_1, E_1', \ldots, E_j, E_j'), \{c_1 = 1, c_2, \cdots, c_j\})$: The prover does:
    1. $b \leftarrow \mathbb{Z}_N$; Set $\hat{F}_0 \leftarrow [b]F_0$.
    2. For $i = 1, \ldots, j$ compute $\hat{E}_i \leftarrow [c_i b]E_i$. Output $(\hat{F}_0, \hat{E}_1, \ldots, \hat{E}_j)$.
ZK.$V_1(F_0, F_0', \hat{F}_0, E_1, E_1', \hat{E}_1, \ldots, E_j, E_j', \hat{E}_j)$: The verifier acts as below:
    1. If $E_i \neq E_0$ for any $i \in \{1, \ldots, j\}$ then sample $d \leftarrow \{0, 1\}$ and output it.
    2. Else sample $d \leftarrow \{-1, 0, 1\}$ and output it.
ZK.$P_2((F_0, F_0', \hat{F}_0, E_1, E_1', \hat{E}_1, \ldots, E_j, E_j', \hat{E}_j), d, s)$: Given $d$, the prover computes
  $r \leftarrow b - d \cdot s \bmod N$, and outputs $r$.
ZK.$V_2((F_0, F_0', \hat{F}_0, E_1, E_1', \hat{E}_1, \ldots, E_j, E_j', \hat{E}_j), \{c_1 = 1, c_2, \cdots c_j\}, d, r)$: The verifier
  returns 1 if all the following checks pass, and otherwise returns 0.
    1. If $d = -1$ return $\left([r]F_0'^t = \hat{F}_0\right) \wedge \bigwedge_{i=1}^{j} \left([c_i r]E_i'^t = \hat{E}_i\right)$.
    2. If $d = 0$ return $\left([r]F_0 = \hat{F}_0\right) \wedge \bigwedge_{i=1}^{j} \left([c_i r]E_i = \hat{E}_i\right)$.
    3. If $d = 1$ return $\left([r]F_0' = \hat{F}_0\right) \wedge \bigwedge_{i=1}^{j} \left([c_i r]E_i' = \hat{E}_i\right)$.
```

Fig. 5. The HVZK Argument for Proving the Commitment and the Well-formedness of Structured Public Keys

present the underlying Σ-protocol in Fig. 5, which is obtained by the conjunction of the basic ID protocol with the *well-formedness* proof for structured public keys, presented in Section 5.1 of [2]. The resulting Σ-protocol can be considered as an extension of the Σ-protocol presented in Figure 7 of [12], to work with *structured public keys.* We also note that since in a structured public key $c_1 = 1$, these two proofs coincide in the cases $j = 0$ and $j = 1$. Similar to [12], we consider two variants of the presented Σ-protocol, one when $F_0 = E_1 = \ldots = E_j = E_0$ which we call the $\mathsf{Special}$ case, and the other when this condition does not hold, which is called the $\mathsf{General}$ case. Next, we prove the security of the presented Σ-protocol.

Theorem 3.3. *The interactive argument in Fig. 5 is correct, has soundness error rate $\frac{1}{2}$ in the* $\mathsf{General}$ *case and soundness error rate $\frac{1}{3}$ in the* $\mathsf{Special}$ *case, and is computational HVZK for the language $\mathbb{L}_j$ assuming GAIP and $\mathbb{C}_j$-VPwAI.*

Proof. The proof is given in the full version of paper [1]. □

Soundness Error Rate. The above theorem showed that the basic interactive argument has soundness error rate 1/2 in the $\mathsf{General}$ case and 1/3 in the $\mathsf{Special}$ case. Therefore, to achieve a target soundness error rate $2^{-\mathsf{sec}}$ for a given security parameter sec, we need to repeat the protocol sec (resp. $\lceil \mathsf{sec} \log_3 2 \rceil$) times.

Making the Protocol Non-interactive. The Σ-protocol in Fig. 5 is an HVZK public coin interactive argument and can be turned into a NIZK argument in the standard manner using a hash function $G : \{0,1\}^* \rightarrow \{0,1\}^{\mathsf{sec}}$ in the $\mathsf{General}$ case, or $G : \{0,1\}^* \rightarrow \{-1,0,1\}^{\lceil \mathsf{sec} \log_3 2 \rceil}$ in the $\mathsf{Special}$ case. Using a 'slow' hash function for G, as in the case of CSI-FiSh, which is 2^h times slower than a normal hash function, we can reduce the number of repetitions to $t_{\mathsf{ZK}}^{\mathsf{General}} = \mathsf{sec} - h$ or $t_{\mathsf{ZK}}^{\mathsf{Special}} = \lceil (\mathsf{sec} - h) \log_3 2 \rceil$, respectively.[2] In the resulting NIZK argument, we denote the prover and verifier by $\mathsf{NIZK}.P$ and $\mathsf{NIZK}.V$.

Both the prover and the verifier need to compute a total of $(j+1)t_{\mathsf{ZK}}$ group actions throughout this protocol, ignoring the cost of the other operations, as they are negligible in comparison to group action computations. The output size of the proof is composed of the hash output and the responses. The former has a size of approximately $t_{\mathsf{ZK}}^{\mathsf{General}}$ bits ($\log_2 3^{t_{\mathsf{ZK}}^{\mathsf{Special}}} \approx t_{\mathsf{ZK}}^{\mathsf{General}}$), while the latter consists of t_{ZK} elements from $\mathbb{Z}_N$, depending on either the $\mathsf{Special}$ or the $\mathsf{General}$ case. It is interesting to note that the proof size does not depend on j.

Lemma 3.1. *The two algorithms* $\mathsf{NIZK}.P$ *and* $\mathsf{NIZK}.V$ *constitute a non-interactive zero-knowledge quantum proof of knowledge in the quantum random oracle model.*

Proof. Since the group action is free [8,11], our schemes have superlogarithmic collision entropy as defined in [38] and unique responses as defined in [18]. Using

[2] As an example, we can choose $h = 16$ for $\mathsf{sec} = 128$ as is done in [2] and [4]. This gives $t_{\mathsf{ZK}}^{\mathsf{General}} = 112$ for the $\mathsf{General}$ case and $t_{\mathsf{ZK}}^{\mathsf{Special}} = 71$ for the $\mathsf{Special}$ case.

the results from Theorem 3.3, [38] implies ZK against quantum attackers while [18], along with a challenge space superpolynomial in sec implies that our protocol is a quantum proof of knowledge. □

4 Key Generation Based on CSI-FiSh

In this section, we review the current isogeny-based key generation protocols for generating the public key of CSI-FiSh. In particular, we look at the key generation of two threshold signature schemes presented in [12,17], and a Distributed Key Generation (DKG) protocol presented in [3]. We also discuss their extension to larger public keys in the same manner as is needed to extend e.g. CSI-FiSh [4].

We also show that the actively secure threshold signature scheme Sashimi proposed by Cozzo and Smart [12], in its general form for Replicated Secret Sharing (RSS), has a security flaw in the key generation. We present a sample attack that ends up giving an honest party a wrong share after the key generation step, without the party realizing. As such, the honest party is rendered unable to sign, resulting in incorrect signatures, even if the signing protocol is correctly executed. We will therefore only focus on the full-threshold version of Sashimi, which is immune to the described attack.

KeyGen: Given E_0, a TTP acts as follows:

1. For $i = 1, \ldots, k-1$, sample secrets $s_i \leftarrow \mathbb{Z}_N$ and use Shamir secret sharing to split the shares s_i into subshares $s_{i,j}$ for $j = 1, \ldots, n$.
2. Distribute $s_{1,j}, \ldots, s_{k-1,j}$ privately to party P_j.
3. Output the public key $\mathsf{pk} := (E_0, \ldots, E_{k-1})$, where $E_i = [s_i]E_0$.

Fig. 6. The key generation protocol of De Feo and Meyer's TSS [17].

At the end, we also discuss some optimizations to the DKG protocols, which allow parties to stagger the computations so as to minimize the idle time in the distributed protocol and thus optimize the overall runtime. This approach was briefly mentioned in [17] and we show that it can also be used in the actively secure case by staggering the zero-knowledge proofs needed for active security. We end each subsection by giving the optimal runtime for n parties by using this type of public key extension and also give an estimate of the communication costs between the parties. The sequential runtimes are simply expressed in terms of group actions. Regarding the communication costs, we note that elliptic curves over finite fields in the CSIDH setting can be expressed with a single parameter of size $\log p$. As $N = \#Cl(\mathcal{O}) \approx \sqrt{p}$, elements in $\mathbb{Z}_N$ can be expressed with approximately $\frac{1}{2}\log p$ bits. Finally, we choose $\mathsf{sec} = \frac{1}{4}\log p$.[3]

[3] Choosing the security parameter this high is meant to reflect the classical security of CSIDH-based protocols against meet-in-the-middle attacks, cf. [8] and sources therein for more details.

4.1 Key Generation of a Passively Secure TSS

De Feo and Meyer [17] presented the first TSS based on isogenies using Shamir secret sharing. Their TSS uses CSI-FiSh to generate distributed signatures and is proven to be secure against passive adversaries. The key generation step is done by a TTP called the *dealer*, who generates a secret key and splits it into shares using Shamir secret sharing (SSS). These shares are distributed securely to the parties and the public key is also computed by the dealer. In Fig. 6, we present this protocol in the case where we have to generate $k-1$ secret elements as for the case of an extended public key. At the end of this protocol, each party P_j will hold a share $s_{i,j}$ of each secret key s_i. We also present the description of the signing protocol in the full version of paper [1].

Cost. It is clear that the TTP has to generate $k-1$ secrets and distribute a total of $n(k-1)$ subshares to the different players. The TTP is left with creating the public key through the computation of $k-1$ group actions.

4.2 Full-Threshold Sashimi

The Original Sashimi Protocol. While the TSS from the last subsection achieves passive security and requires a TTP to perform the key generation,

```
Input: The fixed elliptic curve $E_0$, a set of parties $Q$, a secret shared element $s \in \mathbb{Z}_N$
    held via a full threshold sharing, i.e. $P \in Q$ holds $s_P$ such that $s = \sum_{P \in Q} s_P$.
Output: $[s]E_0$
 1. Define an ordering the players in $Q = \{P_1, \ldots, P_t\}$.
 2. Each party $P_j$ initialises an instance of $\mathcal{F}_{\mathsf{Commit}}$; call it $\mathcal{F}_{\mathsf{Commit}}^{P_j}$.
 3. For $j = 1, \ldots, t$
      - $E_{P_j} \leftarrow [s_{P_j}]E_0$; $\pi_{P_j}^1 \leftarrow \mathsf{NIZK}.P((E_0, E_{P_j}), s_{P_j})$.
      - The parties call $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ where $P_j$ submits input $(\mathsf{Commit}, \mathsf{id}_{P_j}, (E_{P_j}, \pi_{P_j}^1))$
        and all other parties input $(\mathsf{Commit}, \mathsf{id}_{P_j}, \perp)$
 4. For $j = 1, \ldots, t$
      - The parties execute $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ with input $(\mathsf{Open}, \mathsf{id}_{P_j})$ and abort if $\mathcal{F}_{\mathsf{Commit}}^{P_j}$
        returns $\mathsf{abort}$.
      - For all $P_j \neq P_i$ party $P_j$ executes $\mathsf{NIZK}.V((E_0, E_{P_i}), \pi_{P_i}^1)$ and aborts if the
        verification algorithm fails.
 5. $E^0 \leftarrow E_0$.
 6. For $j = 1, \ldots, t$ do
      - Party $P_j$ computes $E^j \leftarrow [s_{P_j}]E^{j-1}$
      - $\pi_{P_j}^2 \leftarrow \mathsf{NIZK}.P((E_0, E_{P_j}, E^{j-1}, E^j), s_{P_j})$.
      - Broadcast $(E^j, \pi_{P_j}^2)$ to all players.
      - All players execute $\mathsf{NIZK}.V((E_0, E_{P_j}, E^{j-1}, E^j), \pi_{P_j}^2)$ and abort if the veri-
        fication algorithm fails.
 7. Return $E^t$.
```

Fig. 7. Group Action Computation for a Full Threshold Secret Sharing [12].

Sashimi [12] aims to achieve active security and uses a Pseudo-Random Secret Sharing (PRSS) to generate a replicated secret sharing. Figure 8 describes the algorithms underlying Sashimi, as presented in [12], using Fig. 7 as a subroutine. During the key generation the parties first run an instance of $\mathcal{F}_{\mathsf{Rand}}.\mathsf{PRSS}()$ and form a secret sharing $\langle a_i \rangle$ of each secret key a_i (we refer the reader to the original paper for more details on $\mathcal{F}_{\mathsf{Rand}}.\mathsf{PRSS}()$ and the protocol that implements it). For RSS, each party holds multiple shares and the same share belongs to multiple parties (except for the special case of full-threshold where each party holds exactly one share of the secret). Then the parties agree on a qualified set Q and turn the RSS $\langle a_i \rangle$ into a full-threshold secret sharing over Q by properly re-arranging the shares and adding them together. This means that each party $P \in Q$ holds a single share $a_{i,P}$ for each key, which is the sum of some (possibly all) the previous shares. The formal way to pass from a replicated to full-threshold is explained in the original paper (Section 2.2 of [12]).

After the sharing phase the parties engage in the $\mathsf{GrpAction}$ protocol for generating the public key elements $E_1, \ldots, E_{k-1}$. Within $\mathsf{GrpAction}$, each party in Q first commits to its shares $a_{i,P}$ and attaches a proof of knowledge. Then it commits to this data using a RO-based commitment scheme and sends the commitment to the other parties. After the successful verification of the proofs, the parties start a round-robin protocol and compute the public keys E_i by using their committed secret shares. Again, the parties give a proof to ensure that they are updating the public key using the same value they committed to earlier.

```
KeyGen: To generate a distributed key we execute:
    1. Call $\mathcal{F}_{\mathsf{Rand}}.\mathsf{Init}()$.
    2. For $i \in [1, \ldots, k-1]$ do
        (a) $\langle a_i \rangle \leftarrow \mathcal{F}_{\mathsf{Rand}}.\mathsf{PRSS}()$.
        (b) $E_i \leftarrow \mathsf{GrpAction}(E_0, Q, \langle a_i \rangle)$ for some qualified set $Q$. If this protocol
            aborts, then abort.
    3. Output $\langle a_1 \rangle, \ldots, \langle a_{k-1} \rangle$ and $E_1, \ldots, E_{k-1}$.
Sign($m, \langle s \rangle$): For a set of qualified parties $Q$ to sign a message $m$ they execute:
    1. Write $Q = \{P_1, \ldots, P_t\} \subset \mathcal{P}$.
    2. For $i = 1, \ldots, t_{\mathsf{S}}$
        (a) Party $P_j$ generates $b_{i,j} \leftarrow \mathbb{Z}_N$, so as to form a full threshold sharing $[b_i]$
            over the $t$ parties.
        (b) The parties execute $E'_i \leftarrow \mathsf{GrpAction}(E_0, Q, [b_i])$.
    3. The parties locally compute $(c_1, \ldots, c_{t_{\mathsf{S}}}) \leftarrow H(E'_1 \| \ldots \| E'_{t_{\mathsf{S}}} \| m)$.
    4. For $i = 1, \ldots, t_{\mathsf{S}}$ party $P_j$ computes $r_{i,j} \leftarrow b_{i,j} - \mathsf{sign}(c_i) \cdot \sum_{\Psi_Q(B)=P_j} a_{|c_i|,B}$.
    5. The parties broadcast their values $r_{i,j}$ and locally compute $r_i \leftarrow \sum_{j=1}^{t} r_{i,j}$.
    6. Output $\{(r_i, c_i)\}_{i=1}^{t_{\mathsf{S}}}$.
```

Fig. 8. The Distributed Key Generation and Signing Protocols of Sashimi [12].

The NIZK proofs inside GrpAction protocol are given for the language

$$\mathbb{L}_j := \left\{ \left((F_0, F_0', E_1, E_1', \ldots, E_j, E_j'),\ s\right) : (F_0' = [s]F_0) \bigwedge \left(\bigwedge_{i=1}^{j} (E_i' = [s]E_i) \right) \right\},$$

for the cases that $j = 0$ and $j = 1$. For these two cases, this language exactly coincides with the language introduced in Eq. (1). As a result, the protocol presented in Fig. 5 also coincides with the one proposed in [12], when $j = 0, 1$.

A Security Flaw in Sashimi. Unfortunately, the ZK proofs inside GrpAction are not enough to guarantee that the parties follow the protocol. A malicious party can deviate from the key generation protocol in such a way, that another (honest) party might not be able to sign anymore and such that after the key generation, the produced signature will not pass the final verification, even if all parties behave honestly in the signing protocol.

The issue arises from the fact that in the key generation phase, after obtaining the secret shares $\langle a_i \rangle$ from the PRSS protocol, each party in Q has to commit to $a_{i,P}$ which is the sum of some of its replicated shares. In particular it might not commit to the single shares they got from the PRSS. Malicious parties could simply commit to a different secret value than their original secret share and generate a NIZK proof and pass the verification, thus effectively changing the "correct" share. Note that this problem only regards the adversarial shares that are also held by an honest party. The full-threshold case, where each party only holds a single share of the secret, is not affected by this issue. As a possible countermeasure, in the public key generation the parties in Q need to commit to all the shares they got from the PRSS and consistency checks need to be done by *all* the parties, not just those in Q.

A Sample Attack. Suppose that we have a $(3, 2)$-threshold access structure done with a RSS scheme. Then, we have a secret x defined as $x = x_1 + x_2 + x_3$, such that P_1 obtains $\{x_2, x_3\}$, P_2 obtains $\{x_1, x_3\}$, and P_3 gets $\{x_1, x_2\}$. Assume wlog that P_1 is the corrupt party and that the qualified set in the key generation step is $Q = \{P_1, P_2\}$. Suppose that the parties agree on re-arranging the shares so that in the GrpAction protocol P_1 enters $x_{P_1} := x_2 + x_3$ and P_2 enters $x_{P_2} := x_1$. If P_1 now enters an arbitrary value y instead of x_{P_1}, then this would not be detected. This is because P_1 only commits to y and not to x_2 and x_3 individually, therefore there is no way for P_2 and P_3 to check this. As a result, the final secret key would be $x' = x_1 + y$ rather than $x = x_1 + x_2 + x_3$, so the final public key would be $E_i' = [x_1 + y]E_0$ instead of $E_i = [x_1 + x_2 + x_3]E_0$. As a result, in the threshold signing protocol, even if both parties of a qualified set $\{P_2, P_3\}$ behave honestly and follow the protocol, the resulting signature cannot be successfully verified under the public key E_i', as it is signed with the secret key $x = x_1 + x_2 + x_3$.

In current design of the protocol, this attack cannot be detected by the other parties, because there is no check inside the GrpAction protocol to ensure that the parties commit to the same secret shares sampled in the secret-sharing phase.

An Actively Secure Full-TSS from Sashimi. Here we present a slightly modified, simpler version of Sashimi for the special case of full-threshold access structure. The signing protocol of the full-threshold version is same as original scheme (shown in Fig. 8), while the full-threshold key generation protocol is described in Fig. 9. The new key generation protocol is a *full-threshold k-MT-GAIP generation protocol*, since it can generally be used to sample a k length MT-GAIP instance in a full-threshold manner. The security proof of the full-threshold version follows in Sashimi [12] but is specialized to the full-threshold case. For completeness we give the proof in the full version of paper [1]. Next, we discuss the efficiency of the resulting full-threshold DKG protocol.

Cost. We express the protocol cost in terms of Group Actions (GAs) and consider the other computational costs as negligible in comparison.

In step 4, all parties compute $k-1$ GAs, $k-1$ times execute the ZK argument in Fig. 5 with $j = 0$, and then verify each other's proofs in step 5. This can be done by all parties in parallel and results in a total cost of $(k-1)(1 + nt_{\mathsf{ZK}}^{\mathsf{Special}})$ GAs, since we are in the $\mathsf{Special}$ case. In step 7, each party first computes $k-1$ elliptic curves, runs $k-1$ times the ZK argument in Fig. 5 with $j = 1$, and then all other players can verify this proof in parallel. Because of the round-robin

Input: The fixed elliptic curve E_0 and a set Q of n parties.
Output: $([s_1]E_0, \dots, [s_{k-1}]E_0)$

1. Parties individually sample $k-1$ secrets $s_i \in \mathbb{Z}_N$ shared between the parties, where $P_j \in Q$ holds $s_{1,j}, \dots, s_{k-1,j}$ such that $s_i = \sum_{P_j \in Q} s_{i,j}$.
2. Define an ordering the players in $Q = \{P_1, \dots, P_n\}$.
3. Each party P_j initialises an instance of $\mathcal{F}_{\mathsf{Commit}}$; call it $\mathcal{F}_{\mathsf{Commit}}^{P_j}$.
4. For $i = 1, \dots, k-1$, each party P_j executes
   - $E_{i,P_j} \leftarrow [s_{i,j}]E_0$.
   - $\pi_{i,j}^1 \leftarrow \mathsf{NIZK}.P((E_0, E_{i,P_j}), s_{i,j})$. `(Run the argument in Figure 5 for` $j=0$`)`
   - Use $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ where P_j submits input $(\mathsf{Commit}, \mathsf{id}_{P_j}, (E_{i,P_j}, \pi_{i,j}^1))$ and all other parties input $(\mathsf{Commit}, \mathsf{id}_{P_j}, \perp)$
5. For $i = 1, \dots, k-1$
   - The parties execute $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ with input $(\mathsf{Open}, \mathsf{id}_{P_j})$ and abort if $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ returns abort.
   - All other players execute $\mathsf{NIZK}.V((E_0, E_{i,P_j}), \pi_{i,j}^1)$ and abort if the verification algorithm fails.
6. $E_1^0 \leftarrow E_0, E_2^0 \leftarrow E_0, \cdots, E_{k-1}^0 \leftarrow E_0$.
7. For $j = 1, \dots, n$
   - Party P_j computes $E_1^j \leftarrow [s_{1,j}]E_1^{j-1}, \cdots, E_{k-1}^j \leftarrow [s_{k-1,j}]E_{k-1}^{j-1}$.
   - For $i = 1, \dots, k-1$, compute $\pi_{i,j}^2 \leftarrow \mathsf{NIZK}.P((E_0, E_{i,P_j}, E_i^{j-1}, E_i^j), s_{i,j})$. `(Run the argument in Figure 5 for` $j=1$`)`
   - Broadcast $(E_1^j, E_2^j, \cdots, E_{k-1}^j, \pi_{1,j}^2, \dots, \pi_{k-1,j}^2)$ to all players.
   - For $i = 1, \dots, k-1$, all other players execute $\mathsf{NIZK}.V(E_0, E_{i,P_j}, E_i^{j-1}, E_i^j)$ and abort if the verification algorithm fails.
8. Return $(E_1^n, E_2^n, \dots, E_{k-1}^n) = ([s_1]E_0, [s_2]E_0, \cdots, [s_{k-1}]E_0)$.

Fig. 9. Full-threshold k-MT-GAIP distributed key generation protocol.

structure, this is repeated sequentially for all players, i.e. n times. Note that since $(E_1^0, E_i^1) = (E_0, E_{i,P_j})$, P_1 does not need to compute anything in this step. The full sequential cost of each party's round in step 7 (except for P_1) amounts to $2(k-1)t_{\mathsf{ZK}}^{\mathsf{General}}$ GA for the proof, and another $2(k-1)t_{\mathsf{ZK}}^{\mathsf{General}}$ GA for the verification, which can be done by all other players in parallel. We end up with a total naive cost of $(k-1)(n+nt_{\mathsf{ZK}}^{\mathsf{Special}}+4(n-1)t_{\mathsf{ZK}}^{\mathsf{General}})$ GA for the full protocol. The runtime of step 7 can however be improved by using an idea similar to the staggering approach mentioned in [17], but where the players stagger the NIZK proofs and verifications. The idea is that at step j, party P_j computes its $k-1$ elliptic curves and then builds $\pi_{1,j}$ (while the other players are idle). As soon as this proof is finished, this proof is broadcast and P_j starts computing $\pi_{2,j}$. At the same time, all other players verify $\pi_{1,j}$, which takes the same amount of computational effort as building the proof. This continues until all players have finally verified $\pi_{k-1,j}$ in the last step (where P_j is idle). It is easy to see, that the round of P_j has a total sequential cost $k-1+2kt_{\mathsf{ZK}}^{\mathsf{General}}$ GA. We finally end up with a total sequential cost of

$$n(k-1)(1+t_{\mathsf{ZK}}^{\mathsf{Special}})+(n-1)2kt_{\mathsf{ZK}}^{\mathsf{General}} \tag{2}$$

for the entire protocol.

For the communication cost, we see that both in step 4 and 7, each party publishes $k-1$ elliptic curves and proofs. We can bound the total communication output per player by $(k-1)\,(10+\log p)\,\frac{1}{4}\log p$.[4]

4.3 CSI-RAShi: A Distributed Key Generation Protocol

In [3], Beullens et al. proposed a DKG protocol based on Shamir's secret sharing [31] called CSI-RAShi, that allows a set of parties to generate a public key $E_1 = [a]E_0$ in a distributed manner, with a being their shared secret. Since Pedersen commitments [27] do not exist as such in the isogeny setting, they can not be used for parties to verify the correctness of their shares. As a way out, the authors introduce *Piecewise Verifiable Proofs* (PVP), which allow parties to prove and verify correctness of their collective and individual shares using ZK proofs in a much faster way than the naive approach. PVPs are a major contribution of [3], however we skip their definition here, as our later improvements are mainly related to CSI-RAShi's bottleneck, which are standard ZK proofs needed in the public-key computation step. We also refer to the original source for the details about the complaint-disqualification mechanisms between the players. In Fig. 10, we present CSI-RAShi extended to generating $k-1$ keys. We also call it *Shamir k-MT-GAIP generation protocol*, as it can be used to sample a k length MT-GAIP instance based on Shamir secret sharing. We note that by simply using one key, we recover the original version from [3].

[4] In order get this compact formula, we are ignoring the parameter h of the slow hash function, as well as the gain resulting from the $\mathsf{Special}$ case. As such, this formula represents an upper bound of the total communication per player.

Input: An elliptic curve E_0, a set $\{P_1, \dots, P_n\}$ of n parties.
Output: A public key $([a_1]E_0, \dots, [a_{k-1}]E_0)$

Verifiable Secret Sharing:

1. For $i = 1, \dots, k-1$, each player P_j samples a degree $t-1$ polynomial $f_i^{(j)}(x)$ with coefficients from $\mathbb{Z}_N$, as well as a uniformly random elliptic curve $R_i^{(j)}$ and computes $R_i'^{(j)} = [f_i^{(j)}(0)]R_i^{(j)}$. Then, each player constructs a PVP
$$\pi_i^{(j)} = (\tilde{\pi}_i^{(j)}, \pi_{i,1}^{(j)}, \dots, \pi_{i,n}^{(j)})$$
which includes a main proof $\tilde{\pi}_i^{(i)}$ as well as individual proof pieces $\pi_{i,l}^{(j)}$ for each other player P_l. Finally, each player publishes the main part $((R_i^{(j)}, R_i'^{(j)}), \tilde{\pi}_i^{(j)})$ and sends $(f_i^{(j)}(l), \pi_{i,l}^{(j)})$ privately to P_l. The main proof $\tilde{\pi}_i^{(j)}$ allows to verify the statement $R_i'^{(j)} = [f_i^{(j)}(0)]R_i^{(j)}$, while a proof piece $\pi_j^{(i)}$ allows a player to verify correctness of their share $f_i^{(j)}(l)$.
2. For $i = 1, \dots, k-1$, each player P_j verifies all the proofs $\tilde{\pi}_i^{(l)}$ and $\pi_{i,j}^{(l)}$ of all other players P_l with respect to their statements. Whenever a proof fails, players broadcast complaints and the interaction of the concerned players are scrutinized by the other players. This might result in the disqualification of players that didn't follow the protocol properly (see [3] for more details).
3. In the end, all the honest players agree on the same set of qualified players $\mathcal{Q} \subset \{1, \dots, n\}$. At this point the joint secret keys are implicitly defined as $a_i = \sum_{j \in \mathcal{Q}} f_i^{(j)}(0)$. Each party P_j derives their share of a_i as $a_{i,j} = \sum_{l \in \mathcal{Q}} f_i^{(l)}(j)$.

Computing the Public Key:

Set $F_1^0 \leftarrow E_0, \dots, F_{k-1}^0 \leftarrow E_0$.

4. For $j = 1, \dots, n$, in a round-robin way, P_j computes
$F_1^j \leftarrow [f^{(j)}(0)]F_1^{j-1}, \dots, F_{k-1}^j \leftarrow [f^{(j)}(0)]F_{k-1}^{j-1}$, then builds the proofs
$$\pi_i'^{(j)} \leftarrow \mathsf{NIZK}.P(R_i^{(j)}, R_i'^{(j)}, F_{j-1}^i, F_j^i),$$
by running the argument in Figure 5 for $j = 1$.
5. These proofs are verified by all parties. Whenever a proof fails, parties again scrutinize the interaction and can disqualify malicious players. In the honest majority setting, parties can even reconstruct missing information if needed, so that the public key implicitly defined by point 3.
6. In the end, the parties return their public key
$$(F_1^{|\mathcal{Q}|}, F_2^{|\mathcal{Q}|}, \dots, F_{k-1}^{|\mathcal{Q}|}) = ([a_1]E_0, [a_2]E_0, \dots, [a_{k-1}]E_0)\,.$$

Fig. 10. The DKG Protocol CSI-RAShi [3] for an Extended Public Key.

Cost. The runtime of CSI-RAShi is very similar to the DKG of Sashimi. A major difference is that in the VSS step, the parties first compute $k-1$ elliptic curves $R_i^{(j)}$ and build their proofs using PVPs. The cost of the PVP is dominated by the main proof (e.g. $\tilde{\pi}_i^{(j)}$) which costs $t_{\mathsf{ZK}}^{\mathsf{General}}$. Adding $n-1$ verifications for each such proof, the VSS ends up costing $(k-1)(2 + nt_{\mathsf{ZK}}^{\mathsf{General}})$ sequential group actions. Steps 4 and 5 amount to the same cost as step 6 of the Sashimi group action, i.e. $2(k-1)t_{\mathsf{ZK}}^{\mathsf{General}}$ per proof and again per verification. The difference with the protocol in Fig. 9 is that party P_1 also has to run these proofs. Finally, we end up with a total naive cost of $(k-1)(2+n+5nt_{\mathsf{ZK}}^{\mathsf{General}})$.[5] By again staggering the proofs and verifications, as described in the Sashimi DKG, we can further reduce the sequential cost of steps 4 and 5. Optimally, we end up with a total sequential cost of

$$(n+2)(k-1)+nt_{\mathsf{ZK}}^{\mathsf{General}}(3k-1)\,. \tag{3}$$

For the communication cost, in the first step, each party publishes $2(k-1)$ elliptic curves and $k-1$ main proofs of the PVPs, then sends $k-1$ shares and proof pieces to each of the $n-1$ other players. In step 4, each party further publishes $k-1$ curves and proofs. After some arithmetic,[6] the total communication output per player can be expressed as $\frac{1}{4}\log p(k-1)(8n+17+\log p)$.

5 Key Generation Based on CSI-SharK

Next, we revisit the key generation protocols based on CSI-FiSh (given in Sect. 4) and show, that by using SPKs (as used in CSI-SharK) instead of standard extended public keys (as used in CSI-FiSh), the cost of these DKG protocols is reduced to a lower cost than current optimal cases (discussed in Sect. 4). This gain mainly arises from the fact, that the VSS steps need to be performed for a single key only, and during the DKG the $k-1$ independent NIZK proofs can be replaced with a single proof of Fig. 5 with $j=k-1$. We then discuss optimizations to the sequential cost of the protocol, which allows us to even further reduce the cost of these protocols by optimally splitting these larger proofs into chunks and staggering them. We note that this approach works when using (multiples of) the same secret key in these protocols, which happens in the SPK case.

5.1 Structured Key Generation in a Passively Secure TSS

Figure 11 presents a passively secure DKG protocol, based on the protocol from [17], which uses the SPKs. For the associated signing protocol we refer to [1].

[5] In [3], another optimization is discussed which reduces the dominant term from $5nt_{\mathsf{ZK}}^{\mathsf{General}}$ to $4nt_{\mathsf{ZK}}^{\mathsf{General}}$. This is achievable by parties already starting to create their own proofs in their idle time. This could also be used in Sashimi, but only has a minor effect when combined with staggering. We choose to omit it here for simplicity, given that staggering will reduce the dominant term to $3nt_{\mathsf{ZK}}^{\mathsf{General}}$ in either case.

[6] Using the description in [3], the communication content in a PVP is composed of a main proof of $\mathsf{sec}(4(n+1)+\log N)$ bits and $2(n+1)$ proof shares, each of sec bits, resulting in a total of $\mathsf{sec}(6(n+1)+\log N)$.

Theorem 5.1. *Under the Power-DDHA and $\mathbb{C}_{k-1}$-VPwAI assumptions, the protocol in Fig. 11 is correct and simulatable.*

Proof. The proof is completely analogous to the proof of Theorem 1 in [17]. □

Computational Cost. We can see that the TTP computes pk using $k-1$ group actions. While this is the same number of computations as in Sect. 4.1, the TTP in this case only has to generate a single secret and distribute n shares of these secrets, reducing the communication by a factor $k-1$.

5.2 Structured Sashimi

In Fig. 12, we present a new variant of the full-threshold DKG protocol presented in Fig. 9, that can be used to generate a SPK. Similar to the previous case, as this protocol can be used to sample a $\mathbb{C}_k$-VPwAI instance in a full-threshold fashion, we call it *full-threshold* $\mathbb{C}_k$*-VPwAI generation protocol*. To achieve active security, parties compute NIZK proofs for the language from Eq. (1) for the cases $j = 0$ and $j = k-1$.

In the full version of paper [1], we also introduce a full-threshold signature scheme based on this DKG protocol, which is a variant of the signing algorithm in Sashimi. We further prove the security of the DKG and the signature scheme against active adversaries in the full version [1].

Computational Cost. We can see that steps 5 and 6 of Fig. 12 have a total cost of $1 + nt_{\mathsf{ZK}}^{\mathsf{Special}}$, as all steps can be done in parallel by all players. Step 8 consists of each player computing $k-1$ isogenies and then building a NIZK proof for $j = k-1$, which is then verified in parallel by all other players. Each such step therefore costs $(k-1)(1 + 2t_{\mathsf{ZK}}^{\mathsf{General}})$. The exception is player P_1, which has already computed $E_{P_j} = E_1^1$ and therefore can exclude it from the NIZK proof. Furthermore, the proofs and verification of P_1's step are in the $\mathsf{Special}$ case, so we end up with the total cost of

$$(n-2)t_{\mathsf{ZK}}^{\mathsf{Special}} + (k-1)(n + 2t_{\mathsf{ZK}}^{\mathsf{Special}} + (n-1)2t_{\mathsf{ZK}}^{\mathsf{General}})\,. \tag{4}$$

We note that the dominant term in this equation scales with $2nkt_{\mathsf{ZK}}^{\mathsf{General}}$, which is already lower than the cost of the protocol reviewed in Sect. 4.2.

KeyGen: Given E_0, a TTP acts as follows:

1. Sample a secret $s \leftarrow \mathbb{Z}_N$ and define a (super)exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \ldots, c_{k-1}\}$.
2. Use SSS to split s into subshares s_j for $j = 1, \ldots, n$ and distribute s_j privately to party P_j.
3. Output the public key $\mathsf{pk} := (E_0, \ldots, E_{k-1})$, where $E_i = [c_i s]E_0$.

Fig. 11. Passive distributed key generation protocol with structured public key.

```
Input: The fixed elliptic curve $E_0$ and a set $Q$ of $n$ parties.
Output: $[s]E_0, [c_2 s]E_0, \cdots, [c_{k-1} s]E_0$

1. Parties agree on a super-exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \ldots, c_{k-1}\}$.
2. Parties individually sample a secret $s_j \in \mathbb{Z}_N$, such that $s = \sum_{P_j \in Q} s_j$.
3. Define an ordering the players in $Q = \{P_1, \ldots, P_n\}$.
4. Each party $P_j$ initialises an instance of $\mathcal{F}_{\mathsf{Commit}}$; call it $\mathcal{F}^{P_j}_{\mathsf{Commit}}$.
5. Each party $P_j$ computes
     - $E_{P_j} \leftarrow [s_j]E_0$.
     - $\pi^1_j \leftarrow \mathsf{NIZK}.P((E_0, E_{P_j}), s_j)$.     (Run the argument in Fig. 5 for $j = 0$)
   All parties call $\mathcal{F}^{P_j}_{\mathsf{Commit}}$ where $P_j$ submits input $(\mathsf{Commit}, \mathsf{id}_{P_j}, (E_{P_j}, \pi^1_j))$ and all
   other parties input $(\mathsf{Commit}, \mathsf{id}_{P_j}, \perp)$
6. For $j = 1, \ldots, n$
     - The parties execute $\mathcal{F}^{P_j}_{\mathsf{Commit}}$ with input $(\mathsf{Open}, \mathsf{id}_{P_j})$ and abort if $\mathcal{F}^{P_j}_{\mathsf{Commit}}$
       returns $\mathsf{abort}$.
     - For all $P_i \neq P_j$, party $P_i$ executes $\mathsf{NIZK}.V((E_0, E_{P_j}), \pi^1_j)$ and aborts if the
       verification algorithm fails.
7. $E^0_1 \leftarrow E_0, E^0_2 \leftarrow E_0, \cdots, E^0_{k-1} \leftarrow E_0$.
8. For $j = 1, \ldots, n$ do
     - Party $P_j$ computes
         $E^j_1 \leftarrow [s_j]E^{j-1}_1, E^j_2 \leftarrow [c_2 s_j]E^{j-1}_2, \cdots, E^j_{k-1} \leftarrow [c_{k-1} s_j]E^{j-1}_{k-1}$,
         $\pi^2_j \leftarrow \mathsf{NIZK}.P((E_0, E_{P_j}, E^{j-1}_1, E^j_1, \cdots, E^{j-1}_{k-1}, E^j_{k-1}), \mathbb{C}_{k-1}, s_j)$.
     - Broadcast $(E^j_1, E^j_2, \cdots, E^j_{k-1}, \pi^2_j)$ to all players.
     - All players execute $\mathsf{NIZK}.V((E_0, E_{P_j}, E^{j-1}_1, E^j_1, \ldots, E^{j-1}_{k-1}, E^j_{k-1}), \mathbb{C}_{k-1}, \pi^2_j)$
       and abort if the verification algorithm fails.
9. Return $([s]E_0, [c_2 s]E_0, \cdots, [c_{k-1} s]E_0) = (E^n_1, E^n_2, \cdots, E^n_{k-1})$.
```

Fig. 12. Full-threshold $\mathbb{C}_k$-VPwAI Generation Protocol.

We can however further optimize step 8 for $k > 2$. Currently, each player computes the full proof with $j = k - 1$, while other players wait, then this proof is verified while the prover waits. Instead, we can subdivide the one full proof of $k - 1$ elements into r smaller proofs of $j = \lceil \frac{k-1}{r} \rceil$ elements. The idea is to *stagger* these smaller proofs and verifications steps as was done in Sect. 4.2. While the proving party computes the first proof, the other players are idle. After finishing this proof, it is published, and the proving player computes its second proof while the other players verify the first one. This is repeated r times. In the last round, the other parties verify the rth proof, while the prover is idle. As such, there is a big overlap between the computations of the prover and the verifiers, reducing the overall idle time. Assuming for simplicity that r divides $k - 1$, it is clear that each of the proofs should contain the pair (E_0, E_{P_j}) as the reference curves, as well as $\frac{k-1}{r}$ other curves, which should be proven to have been computed correctly. Given the staggered approach, we end up with a total of $r+1$ proof-verification cycles, where each costs $\left(\frac{k-1}{r}+1\right) t_{\mathsf{ZK}}$ group actions. By assuming the prover has already precomputed the (E_0, E_{P_j}) part of its first proof, as explained above, we can reduce the cost of the first cycle to $\frac{k-1}{r} t_{\mathsf{ZK}}$ group actions. Including the costs of the elliptic curve computations, this yields a total of $k-1+\frac{k-1}{r} t_{\mathsf{ZK}}+r\left(\frac{k-1}{r}+1\right) t_{\mathsf{ZK}}$ sequential group actions per round. It is clear that this cost is minimized for $r=\sqrt{k-1}$. We end up with a cost per round-robin step of $k-1+\left((\sqrt{k-1}+1)^2-1\right) t_{\mathsf{ZK}}$ for players $P_2, \ldots, P_n$.

We are left with establishing the cost of P_1's round. Again, P_1 only needs to compute $k-2$ curves and consequently prove correctness of these $k-2$ elements. Using the same process as before we end up with an optimal $r'=\sqrt{k-2}$, i.e. we have a total of $\sqrt{k-2}+1$ cycles with proofs/verifications of cost $(\sqrt{k-2}+1) t_{\mathsf{ZK}}^{\mathsf{Special}}$ per cycle. Adding the cost of step 5, we end up with the total sequential cost of the DKG protocol of:[7]

$$\begin{aligned} T^{DKG}(n, k, \mathsf{sec}) = n(k-1) &+ \left(n+(\sqrt{k-2}+1)^2\right) t_{\mathsf{ZK}}^{\mathsf{Special}} \\ &+ (n-1)\left((\sqrt{k-1}+1)^2-1\right) t_{\mathsf{ZK}}^{\mathsf{General}}. \end{aligned} \tag{5}$$

The cost in Eq. (4) is dominated by the term $2knt_{\mathsf{ZK}}^{\mathsf{General}}$, while here, this term is reduced to $n(k+2\sqrt{k}) t_{\mathsf{ZK}}^{\mathsf{General}}$. For large k, this gives another improvement by almost a factor of 2.

It is easy to see that our protocol also strongly reduces the communication cost needed between the parties. First of all, by using only one secret instead of $k-1$, we reduce the communication cost of the VSS step by a factor $k-1$. Similarly, in the public-key computation step, we reduce the number of proofs from $k-1$ to $\sqrt{k-1}$ per player, and even to 1 in the non-optimized case. Only the number of elliptic curves published in the public-key computation step stays constant, when compared to the non-structured case. Expressed in bits, we find $\frac{1}{4}\log p\,(4k+7n+7+\log p)$ in the case with one single proof and $\frac{1}{4}\log p\left(4k+7n+6+\frac{1}{2}\log p+(\sqrt{k-1}+1)(1+\frac{1}{2}\log p)\right)$ in the optimized case, as the communication output per player. It is easy to see that asymptotically for large k, both these cases yield a gain factor of $\frac{1}{4}(10+\log p)$, when compared to the protocol in Sect. 4.2.

[7] Note that Eq. (5) is slightly simplified, since in general, the square roots within this equation are not integers. If e.g. $\sqrt{k-1}$ is not an integer, we construct $r=\lceil\sqrt{k-1}\rceil$ proofs, each of size at most $\lceil\frac{x}{r}\rceil$. This means that we can substitute the terms of type $(\sqrt{x}+1)^2$ by the term $(\lceil\sqrt{x}\rceil+1)(\lceil\frac{x}{\lceil\sqrt{x}\rceil}\rceil+1)$ (where e.g. $x \in \{k-1, k-2\}$) in order to upper bound the actual cost.

Input: An elliptic curve E_0, a set $\{P_1, \ldots, P_n\}$ of n parties.
Output: A public key $([a]E_0, [c_2 a]E_0, \cdots, [c_{k-1}a]E_0)$
Verifiable Secret Sharing:

1. Each player P_i, for $i = 1, \ldots, n$, samples a degree $t-1$ polynomial $f^{(i)}(x)$ with coefficients from $\mathbb{Z}_N$, as well as a uniformly random elliptic curve $R^{(i)}$ and computes $R'^{(i)} = [f^{(i)}(0)]R^{(i)}$. Then, each player constructs a PVP
$$\pi^{(i)} = (\tilde{\pi}^{(i)}, \pi_1^{(i)}, \ldots, \pi_n^{(i)})$$
which includes a main proof $\tilde{\pi}^{(i)}$ as well as individual proof pieces $\pi_j^{(i)}$ for each other player P_j. Finally, each player publishes the main part $((R^{(i)}, R'^{(i)}), \tilde{\pi}^{(i)})$ and sends $(f^{(i)}(j), \pi_j^{(i)})$ privately to P_j. The main proof $\tilde{\pi}^{(i)}$ allows verifying the statement $R'^{(i)} = [f^{(i)}(0)]R^{(i)}$, while a proof piece $\pi_j^{(i)}$ allows a player to verify the correctness of their share $f^{(i)}(j)$.
2. Now, each player P_j verifies all the proofs $\tilde{\pi}^{(i)}$ and $\pi_j^{(i)}$ of all other players with respect to their statements. Whenever a proof fails, players broadcast complaints and the interaction of the concerned players are scrutinized by the other players. This might result in the disqualification of players that didn't follow the protocol.
3. In the end, all the honest players agree on the same set of qualified players $\mathcal{Q} \subset \{1, \ldots, n\}$. At this point the joint secret key is implicitly defined as $a = \sum_{i \in \mathcal{Q}} f^{(i)}(0)$. Each party P_j derives their share of a as $a_j = \sum_{i \in \mathcal{Q}} f^{(i)}(j)$.

Computing the *Structured* Public Key:

4. Parties agree on a super-exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \ldots, c_{k-1}\}$.
5. In a round-robin way, the qualified players compute
$$F_i^1 \leftarrow [f^{(i)}(0)]F_{i-1}^1, F_i^2 \leftarrow [c_2 f^{(i)}(0)]F_{i-1}^2, \ldots, F_i^{k-1} \leftarrow [c_{k-1} f^{(i)}(0)]F_{i-1}^{k-1},$$
where $F_0^1 = \cdots = F_0^{k-1} = E_0$. At each step, player P_i publishes the proof
$$\pi'^{(i)} \leftarrow \mathsf{NIZK}.P((R^{(i)}, R'^{(i)}, F_{i-1}^1, F_i^1, \ldots, F_{i-1}^{k-1}, F_i^{k-1}, \mathbb{C}_{k-1}), f^{(i)}(0)),$$
by running the NIZK argument in Fig. 5 for $j = k-1$.
6. This proof is verified by all parties. Whenever a proof fails, parties again scrutinize the interaction and can disqualify malicious players. In the honest majority setting, parties can even reconstruct missing information if needed, so that the public key is implicitly defined by point 3.
7. In the end, the parties return their structured public key
$$(F_{|\mathcal{Q}|}^1, F_{|\mathcal{Q}|}^2, \ldots, F_{|\mathcal{Q}|}^{k-1}) = ([a]E_0, [c_2 a]E_0, \ldots, [c_{k-1}a]E_0)\,.$$

Fig. 13. Shamir $\mathbb{C}_k$-VPwAI Generation Protocol.

5.3 Structured CSI-RAShi

We conclude this section by presenting a CSI-RAShi-based DKG protocol for an SPK in Fig. 13. The protocol is called *Shamir $\mathbb{C}_k$-VPwAI generation protocol*, as it can also be used to sample a $\mathbb{C}_k$-VPwAI instance in a fully distributed manner using Shamir secret sharing. We also present an actively secure TSS based this DKG in the full version of paper [1].

Theorem 5.2. *The protocol of Fig. 13 satisfies the consistency requirement, assuming GAIP and $\mathbb{C}_{k-1}$-VPwAI and satisfies the secrecy requirement, further assuming the $\mathbb{C}_{k-1}$-Decisional GAIP.*

Proof. The proof is given in the full version of paper [1]. □

Computational Cost. In contrast to the extension with multiple secret keys, the SPK extension in Fig. 13 requires only *a single execution* of the VSS step, instead of $k-1$ repetitions, and so it is independent of the public key size. The cost of this step is $2 + nt_{\mathsf{ZK}}^{\mathsf{General}}$ group actions. Furthermore, parties do not have to repeat separate ZK proofs multiple times, but can rather compute one big proof for all $k-1$ elements. This results in a cost per round-robin step of $(k-1)(1+2t_{\mathsf{ZK}}^{\mathsf{General}})$, resulting in the total sequential cost of

$$2 + nt_{\mathsf{ZK}}^{\mathsf{General}} + n(k-1)(1+2t_{\mathsf{ZK}}^{\mathsf{General}}) \tag{6}$$

for the entire protocol. It is easy to see that the dominant term scales with $2nkt_{\mathsf{ZK}}^{\mathsf{General}}$, which is lower than the cost of the protocol in Sect. 4.3. We can improve this cost by using the staggering approach described in Sect. 5.2, in which we split the main proof into $\sqrt{k-1}$ smaller proofs of size approximately $\sqrt{k-1}$. A difference to the cost established in that section is again that P_1 is not in the $\mathsf{Special}$ case and has to compute $k-1$ curves (as opposed to $k-2$ in Fig. 12). We end up with the final cost

$$2 + n(k-1) + t_{\mathsf{ZK}}^{\mathsf{General}}\left(n(\sqrt{k-1}+1)^2 + 1\right). \tag{7}$$

The dominating term in CSI-RAShi scales as $3nkt_{\mathsf{ZK}}^{\mathsf{General}}$, while using structured keys can reduce this to $n(k+2\sqrt{k})t_{\mathsf{ZK}}^{\mathsf{General}}$. For large k, this gives an improvement of almost a factor 3.

Again, we note that our scheme reduces the communication cost in the VSS by a factor $k-1$, as only a single PVP is needed. Similarly, in the public-key computation step, we reduce the number of proofs from $k-1$ to $\sqrt{k-1}$ per player, or 1 in the non-optimized case, while the number of published curves stays $k-1$. For the case with a single proof, we find the total cost of $\frac{1}{4}\log p\,(4k+6n+10+\log p)$, while in the optimized case, we find $\frac{1}{4}\log p(4k+6n+9+(\sqrt{k-1}+1)(1+\frac{1}{2}\log p))$, as the communication output per player.

For asymptotically large k, in comparison to the non-structured case, we get a gain factor of the communication of $\frac{1}{4}(8n+17+\log p)$, when compared to Sect. 4.3. We note that the gain increases with the number of parties. This is due to the fact that the size of the PVPs depends on the number of players.

6 Parallel Executions and Performances

Next, we discuss the benefits of parties having multiple CPU threads at their disposal when executing isogeny-based signatures and NIZK proofs. Given the fact that multi-core CPUs are standard, we want to consider their impact on the schemes presented in this work. As observed in all threshold protocols, to compute a curve $[x]E_0$, where x is shared among multiple parties, having to adopt a sequential round-robin communication structure between the parties seems to be an unavoidable fact. However, we observe that in the cases that we need to compute more than one curve, e.g. $[x_1]E_0, \ldots, [x_{k-1}]E_0$, be it either as a single party (as in the CSI-FiSh or CSI-SharK) or by several parties (as in the threshold variants of them or DKG protocols), these computations are in general independent of each other and can therefore very easily be parallelized.

In the case of *non-threshold* protocols, where only one party runs the algorithm, the parallelization can be done using different threads (or cores) of a CPU. Namely, each thread of the CPU can compute a subset of these curves, and will finish the total computation considerably faster than the consecutive approach described in the original CSI-FiSh [4] and its follow-up schemes [3,12]. As an example, assuming a party has C independently accessible threads, they can compute a CSI-SharK (or CSI-FiSh) signature for the consecutive cost of $\lceil t_S/C \rceil$ group actions instead of the full t_S, because of the independence of the commitments. In the full version of paper [1], we show the impact of parallelizing signature schemes in this way. We observe that using 8 cores with a public key of size $k = 2$ (64 B) already allows for signing and verifying as fast as using a public key of size $k = 2^{12} + 1$ (256 KB) with a single core.

In the case of *threshold* protocols, the same idea can be applied to NIZK arguments, such as the one in Fig. 5, reducing the proof cost from $(j+1)t_{\mathsf{ZK}}$ to $(j+1)\lceil t_{\mathsf{ZK}}/C \rceil$ group actions. Since all commitments can be independently computed, and the j individual commitments are parallelizable, we can further reduce this to $\lceil (j+1)t_{\mathsf{ZK}}/C \rceil$ group actions, effectively reducing the sequential costs of our algorithms by a factor C, up to some constant terms. Furthermore, while parties are still forced to adopt a sequential structure due to the round robins, they can however run multiple such round-robins in parallel to fill up the idle time. This was already observed in [17] for passively secure threshold protocols. Here we extend this idea to actively secure threshold case, where parties also need to generate ZK proofs and verify the proofs of other parties.

In Table 2, we present estimates of the performances of our actively secure threshold signature schemes based on CSI-SharK for different parameter sets. We compare our schemes with the DKGs from [3,12] and the signature scheme from [12]. For these estimates, we use the formulas established throughout this work, including the results for signatures from [1]. We compare them to the original time complexity formulas from the relevant sources. We use the (conservative) approximate cost of 40 ms per group action that can be expected when combining the benchmarks presented in [4] with the optimizations from [26] on a 3.5 GHz processor [8]. The table indicates that SPKs give a strong performance benefit, even when compared to the most optimal case without structured keys.

Since multiple threads are standard in modern CPU architecture, our results show that isogeny-based signature and threshold schemes become quite practical by virtue of the parallelizability of their computations. For instance, 3 parties,

Table 2. Comparison of computational and communication cost for different instances of the full-threshold Sashimi DKG (upper table) and CSI-RAShi (lower table) for the CSIDH-512 parameter set. We compare both schemes in the CSI-FiSh (extended public key) and CSI-SharK cases (structured public key) for different public key sizes $k-1$. We compare *naive* implementations (as presented in their original paper or by simply using structured public keys) and optimized (*staggered*) implementations, as discussed in Sect. 4 and Sect. 5. We also indicate the communication cost (output per party) for each of these cases. We note that the communication costs in the CSI-FiSh case do not change when using the staggering optimization, while it does so in the CSI-SharK setting. In the full-threshold case, we further indicate the signature cost, as established in [1]. For completeness, we also compare secret key sizes $|\mathsf{sk}|$ per party in the CSI-FiSh and CSI-SharK cases in the lower table. Runtimes are estimated based on the conservative estimate that a group action computation takes 40 ms. For better readability, some communication cost and secret key size entries are omitted in the tables, as they are simply the same results as in the block above.

Full-		CSI-FiSh			CSI-SharK				
Threshold	$k-1$	naive	staggered	comm.	naive		staggered		Signing
Case		[12]		(Sec. 4)	(Sec. 5)		(Sec. 5)		App. of [1]
3 parties	2^4	17 min	10 min	131 kB	7.3 min	9.1 kB	5.5 min	21 kB	12 min
each with	2^8	4.5 h	2.6 h	2.0 MB	1.9 h	24 kB	65 min	84 kB	7.2 min
1 core	2^{12}	73 h	42 h	33 MB	31 h	264 kB	16 h	517 kB	4.5 min
3 parties	2^4	67 s	38 s		27 s		21 s		47 s
each with	2^8	18 min	10 min		7 min		4 min		28 s
16 cores	2^{12}	4.8 h	2.6 h		1.9 h		60 min		18 s
8 parties	2^4	3.1 min	1.8 min		80 s		60 s		147 s
each with	2^8	50 min	28 min		21 min		12 min		94 s
16 cores	2^{12}	13 h	7.4 h		5.6 h		2.9 h		87 s

Shamir		CSI-FiSh				CSI-SharK				
secret	$k-1$	naive	staggered	comm.	$\lvert\mathsf{sk}\rvert$	naive		staggered		$\lvert\mathsf{sk}\rvert$
sharing		[3]		(Sec. 4)		(Sec. 5)		(Sec. 5)		
3 parties	2^4	18 min	13 min	138 kB	512 B	8.5 min	9.5 kB	6.5 min	26 kB	32 B
each with	2^8	4.7 h	3.3 h	2.1 MB	8 kB	2.2 h	25 kB	65 min	89 kB	32 B
1 core	2^{12}	76 h	53 h	34 MB	128 kB	35 h	265 kB	18 h	522 kB	32 B
3 parties	2^4	69 s	48 s			32 s		24 s		
each with	2^8	18 min	13 min			8.2 min		4.7 min		
16 cores	2^{12}	4.9 h	3.3 h			2.2 h		68 min		
8 parties	2^4	2.9 min	2.1 min	148 kB		85 s	10 kB	65 s	26 kB	
each with	2^8	47 min	33 min	2.3 MB		22 min	25 kB	12 min	89 kB	
16 cores	2^{12}	12 h	8.8 h	36 MB		5.8 h	265 kB	3.0 h	522 kB	

each with 16 cores, could sign in 28 s by using a 16 KB large public key, while 8 parties would need 94 s with the same parameters. Key generation in these settings can be done in a few minutes and verification will take 40 ms. We note that by using the optimizations from the full version [1], the runtime of our signature is about twice as fast as the naive approach presented in [12]. This is independent of the public key structure and is readily applicable to signature schemes without structured public keys as well, such as [12].

7 Conclusion

We presented CSI-SharK as a new variant of CSI-FiSh [4], that allows one to build more efficient distributed protocols than those based on CSI-FiSh. CSI-SharK is based on a modified version of the Σ-protocol underlying CSI-FiSh [4]. A *key* difference between CSI-SharK and CSI-FiSh is that CSI-SharK uses SPKs, as recently introduced in [2]. At the cost of an additional computational assumption, CSI-SharK improves CSI-FiSh in a number of ways. Public keys with $k-1$ elliptic curves are now generated using a single secret, instead of $k-1$, which means that only one secret needs to be generated and stored, independent of the public key size. This is most noticeable in distributed key generation protocols, where secrets are further split into parts and distributed among the parties. The heavy zero-knowledge proofs in current state-of-the-art protocols can be strongly reduced in numbers and even merged, when working with structured public keys. This reduces even the most optimal implementations by a factor 3 and the communication cost by a much greater factor, e.g. up to 130 for Sashimi and about $132 + \frac{3}{4}n$ for CSI-RAShi, where n is the number of parties.

While these results make threshold schemes in the isogeny setting much more practical, we further presented a general strategy for parallel computations, exploiting the independence of commitments in zero-knowledge proofs and signature schemes. We show that by parallelizing computations, isogeny-based threshold (and non-threshold) signatures become truly practical and competitive in the post-quantum realm.

As an independent contribution, we revealed a flaw in the DKG protocol of Sashimi, which can allow an honest party to end up with a wrong share after the protocol, thus preventing it from generating correct signatures, even after a correctly executed signing protocol.

We think that the very design of the CSI-SharK public keys, its Σ-protocol and some of our proposed structured DKG protocols can offer many advantages for different applications and hope that the structure may be further exploited to design more practical isogeny-based cryptographic primitives.

Acknowledgments. We would like to thank Prof. Nigel Smart for his valuable comments. This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. HR001120C0085, by the FWO under an Odysseus project GOH9718N, by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101020788 - Adv-ERC-ISOCRYPT), by CyberSecurity Research Flanders with reference number VR20192203, by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under project PRODIGY (TED2021-132464B-I00), and by the Madrid Regional Government under project BLOQUES (S2018/TCS-4339). The last two projects are co-funded by European Union EIE, and Next Generation EU/PRTR funds.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC, DARPA, the US Government, the Spanish Government, Cyber Security Research

Flanders or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

1. Atapoor, S., Baghery, K., Cozzo, D., Pedersen, R.: CSI-SharK: CSI-FiSh with sharing-friendly keys. Cryptology ePrint Archive, Report 2022/1189 (2022). https://eprint.iacr.org/2022/1189
2. Baghery, K., Cozzo, D., Pedersen, R.: An isogeny-based ID protocol using structured public keys. In: Paterson, M.B. (ed.) IMACC 2021. LNCS, vol. 13129, pp. 179–197. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92641-0_9
3. Beullens, W., Disson, L., Pedersen, R., Vercauteren, F.: CSI-RAShi: distributed key generation for CSIDH. In: Cheon, J.H., Tillich, J.-P. (eds.) PQCrypto 2021 2021. LNCS, vol. 12841, pp. 257–276. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81293-5_14
4. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 227–247. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_9
5. Bishnoi, A., Clark, P.L., Potukuchi, A., Schmitt, J.R.: On zeros of a polynomial in a finite grid. Comb. Probab. Comput. **27**(3), 310–333 (2018)
6. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 493–522. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_17
7. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version) (2022). https://ia.cr/2022/975
8. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
9. Chávez-Saab, J., Chi-Domínguez, J.J., Jaques, S., Rodríguez-Henríquez, F.: The Sqale of CSIDH: Square-root Vélu quantum-resistant isogeny action with low exponents. Technical report, Cryptology ePrint Archive, Report 2020/1520, 2020 (2020)
10. Childs, A., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum Subexponential time. J. Math. Cryptol. **8**(1), 1–29 (2014)
11. Couveignes, J.M.: Hard homogeneous spaces. IACR Cryptol. ePrint Arch. **2006**, 291 (2006)
12. Cozzo, D., Smart, N.P.: Sashimi: cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 169–186. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_10
13. Dalskov, A., Lee, E., Soria-Vazquez, E.: Circuit amortization friendly Encodingsand their application to statistically secure multiparty computation. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12493, pp. 213–243. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64840-4_8
14. De Feo, L.: Mathematics of isogeny based cryptography. arXiv preprint: arXiv:1711.04062 (2017)
15. De Feo, L., Galbraith, S.D.: SeaSign: compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 759–789. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_26

16. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: compact post-quantum signatures from quaternions and isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 64–93. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_3
17. De Feo, L., Meyer, M.: Threshold schemes from isogeny assumptions. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12111, pp. 187–212. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_7
18. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Security of the Fiat-Shamir transformation in the quantum random-oracle model. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 356–383. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_13
19. Duman, J., Hartmann, D., Kiltz, E., Kunzweiler, S., Lehmann, J., Riepel, D.: Generic models for group actions. Cryptology ePrint Archive, Report 2023/186 (2023). https://eprint.iacr.org/2023/186
20. El Kaafarani, A., Katsumata, S., Pintore, F.: Lossy CSI-FiSh: efficient signature scheme with tight reduction to decisional CSIDH-512. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12111, pp. 157–186. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_6
21. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
22. Galbraith, S.D., Petit, C., Silva, J.: Identification protocols and signature schemes based on Supersingular isogeny problems. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 3–33. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_1
23. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (2007)
24. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from Supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
25. Maino, L., Martindale, C.: An attack on SIDH with arbitrary starting curve (2022). https://ia.cr/2022/1026
26. Meyer, M., Reith, S.: A faster way to the CSIDH. In: Chakraborty, D., Iwata, T. (eds.) INDOCRYPT 2018. LNCS, vol. 11356, pp. 137–152. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05378-9_8
27. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
28. Peikert, C.: He gives C-sieves on the CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 463–492. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_16
29. Robert, D.: Breaking SIDH in polynomial time (2022). https://ia.cr/2022/1038
30. Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. IACR Cryptol. ePrint Arch. **2006**, 145 (2006)
31. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
32. Shaw, S., Dutta, R.: Identification scheme and forward-secure signature in identity-based setting from isogenies. In: Huang, Q., Yu, Yu. (eds.) ProvSec 2021. LNCS, vol. 13059, pp. 309–326. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90402-9_17

33. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 124–134 (1994)
34. Siegel, C.: Über die classenzahl quadratischer zahlkörper. Acta Arith **1**(1), 83–86 (1935)
35. Silverman, J.H.: The Arithmetic of Elliptic Curves, vol. 106. Springer, Berlin (2009)
36. Stolbunov, A.: Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. Adv. Math. Commun. **4**(2), 215 (2010)
37. Stolbunov, A.: Cryptographic schemes based on isogenies (2012)
38. Unruh, D.: Post-quantum security of Fiat-Shamir. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 65–95. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_3
39. Yoo, Y., Azarderakhsh, R., Jalali, A., Jao, D., Soukharev, V.: A post-quantum digital signature scheme based on supersingular isogenies. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 163–181. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_9

Practical Verifiable Random Function with RKA Security

Tsz Hon Yuen[1(✉)], Shimin Pan[1], Sheng Huang[1], and Xiaoting Zhang[2]

[1] University of Hong Kong, Hong Kong, China
{thyuen,smpan}@cs.hku.hk, vicw0ng@connect.hku.hk
[2] Nanjing University, Nanjing, China

Abstract. A verifiable random function (VRF) allows the generation of a random number with publicly verifiable proof, showing that the random number is honestly generated. The practical VRF used in real-world applications considers the security of uniqueness, pseudorandomness, and unpredictability under malicious key generation. In this paper, we propose the security model of *related-key attack* to VRF for capturing attacks like tampering attacks. We propose a new construction of VRF that satisfies the RKA security together with the existing security requirements. We implement our VRF construction and demonstrate that our scheme is practical for real-world applications.

Keywords: VRF · Verifiable random function · universal composability

1 Introduction

Verifiable random function (VRF), introduced by Micali, Rabin, and Vadhan (FOCS'99), is the public-key equivalent of the pseudorandom function (PRF). A private key sk and a seed value X are passed to a VRF. The VRF outputs a random number $Y = F_{\mathsf{sk}}(X)$ along with a proof π. Anyone can validate if Y is a random number by using a public key pk, the proof π, and the seed X.

VRF is essential in decentralized applications because its tamper-proof ensures that the results cannot be manipulated and thus guarantees fair outcomes. Most random number generators do not produce a random number that can be cryptographically verified, leaving them vulnerable to manipulation and hence limiting their use cases. VRF, however, by guaranteeing the security of random numbers, supports a wide range of uses such as DNSSEC protocol [25], resettable zero-knowledge proof [23], non-interactive lottery systems [8], and consensus algorithm in blockchain [7,9,13,14].

1.1 Applications of VRF

Currently, the most widely adopted VRF is the EC-VRF [25], which is based on elliptic curves. Suppose that the public key $\mathsf{vk} = g^{\mathsf{sk}}$ for some generator g of the

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 503–522, 2023.
https://doi.org/10.1007/978-3-031-35486-1_22

elliptic curve. The VRF outputs $Y = H(X)^{\mathsf{sk}}$, where H is a collision-resistant hash function. The proof π is a classical zero-knowledge proof of the equivalent of discrete logarithm between $\log_g \mathsf{pk}$ and $\log_{H(X)} Y$. EC-VRF was originally proposed for the DNSSEC protocol [25]. EC-VRF is currently being standardized by CRFG [15]. Several layer-1 blockchains, including Algorand [14], Cardano [9] and Polkadot [7], use VRF in their consensus mechanisms to randomly select block producers.

Apart from the consensus algorithm, smart contract developers also require a source of randomness for their applications. The smart contract itself cannot generate a unique random number across different nodes in the distributed network. Chainlink provides smart contracts with a secure implementation of EC-VRF [25], of which the VRF [10] is being used as a secure source of on-chain randomness across the Web3 ecosystem, including GameFi, DeFi, and NFT projects.

The consensus algorithm in DFINITY [17] uses a VRF output that is the hash value of a threshold version of the BLS signature. They consider a case where the secret key sk is distributed among n parties by secret sharing. These parties generate partial BLS signatures. Once the system obtains t shares of partial signatures, the threshold BLS signature can be recovered to calculate Y. The proof π includes t shares of partial BLS signatures, and each partial signature can be validated by a pairing computation.

1.2 Motivation: Stronger Security for VRF

Two security requirements for VRF are defined, namely *uniqueness* and *pseudorandomness*. Since VRF is used in many blockchain applications and affects the security of digital assets worth billions of dollars, VRF is currently defined with a strong security model.

Roughly speaking, the security model of *uniqueness* means that no adversary can output a seed X and two pairs (Y_0, π_0) and (Y_1, π_1) which are both valid with respect to an adversarially chosen verification key vk. The security model of *pseudorandomness* means that an attacker, seeing a number of VRF outputs and proofs for adversarially chosen inputs under an honestly generated key pair, cannot distinguish the output of the VRF on a new (also adversarially chosen) input from a random string. Alternatively, [9] proposed a Universal Composable (UC) security model of VRF with unpredictability under malicious key generation.

Related Key Attack. A related-key attack (RKA) means that an adversary can observe the outcome of a cryptosystem under a modified key (i.e., *related key*) and then breaks the system. RKA attacks can model tampering attacks on the secret key.

The earliest discussion on RKA mainly focused on the security of block ciphers and pseudorandom functions (PRFs) [1,4]. For the RKA-security of PRF, an adversary attempts to break the pseudorandomness of PRF with several secret keys satisfying some known relation. It is natural to ask if it is possible to construct a VRF with security against related key attacks. Since VRF is widely used in blockchain applications, it is realistic to consider its security under tampering attacks.

In this paper, we consider the RKA security of VRF using the formalization of RKA security in the public key cryptosystem. The adversary is allowed to query a *related-key deriving* function ϕ and a seed X to an oracle, and obtain the VRF output $Y' = F_{\phi(\mathsf{sk})}(X)$ and its proof π' of the queried seed X under the related key $\phi(\mathsf{sk})$. The RKA security requires that the adversary cannot break the pseudorandomness even when given access to the oracle.

1.3 High Level Idea

We observe that EC-VRF cannot achieve RKA security for pseudorandomness, with respect to additive relation. Recall that in EC-VRF evaluation with a related secret key $\mathsf{sk} + \varDelta$, the corresponding output of $Y' = H(X)^{\mathsf{sk}+\varDelta}$. The adversary can calculate $Y'/H(X)^{\varDelta}$ and breaks the pseudorandomness. The linear structure of EC-VRF makes it difficult to achieve RKA security for pseudorandomness.

In order to achieve RKA security for pseudorandomness, we first try to modify the pairing-based VRF proposed by Dodis and Yampolskiy [11]:

$$Y = \hat{e}(g, g)^{\frac{1}{\mathsf{sk}+H(X)}}.$$

The corresponding proof is $\pi = g^{\frac{1}{\mathsf{sk}+H(X)}}$ and it can be checked by $Y = \hat{e}(g, \pi)$ and $\hat{e}(\mathsf{pk} \cdot g^{H(X)}, \pi)$. Although it seems that it can achieve the RKA security, [9, section 3.2] mentioned that [11] cannot achieve Universal Composable (UC) security of unpredictability under malicious key generation. An adversary that maliciously generates keys can skew the output distribution.

Alternative, we first try to define

$$U = X^{\frac{1}{\mathsf{sk}}}, \quad Y = H_0(X, U).$$

Define ρ as a NIZK proof for sk such that $U = X^{\frac{1}{\mathsf{sk}}}$ and $\mathsf{vk} = g^{\mathsf{sk}}$. The proof $\pi = (U, \rho)$. We try to achieve the RKA security by the exponent $\frac{1}{\mathsf{sk}}$, and the UC unpredictability by using the hash H_0 as in [9]. Although this scheme seems to be secure, we cannot simulate the evaluation oracle for pseudorandomness without the knowledge of sk.

In order to complete the security proof, we define

$$U = H_1(\mathsf{vk}, X)^{\frac{1}{\mathsf{sk}}}, \quad Y = H_0(X, U).$$

We model the hash function H_1 as a random oracle to simulate the security proof of pseudorandomness. The extra input vk is added to H_1 to handle the simulation of H_1 under different keys (both honestly and maliciously generated keys) in the UC security model. The final obstacle is to have an efficient NIZK proof of sk such that $U = g^{\frac{1}{\mathsf{sk}}}$ and $\mathsf{vk} = g^{\mathsf{sk}}$. We cannot simply use the proof of the equality of discrete logarithm as in the existing schemes. In this paper, we give an efficient proof of inversion relation, inspired from the Bulletproof [6].

1.4 Our Contributions

We define the first RKA security model for VRF and propose the *first* VRF scheme that achieves the standard security requirement of uniqueness, pseudorandomness, and also the RKA security for pseudorandomness. Furthermore, our scheme also has UC security of unpredictability under malicious key generation. Our construction is very practical and the efficiency is close to the existing EC-VRF used in many blockchain applications. Since our scheme provides a higher level of security, it can be applied to many practical applications.

2 Preliminaries

2.1 Verifiable Random Function

We define the verifiable random function (VRF) in this section. A VRF function family $F_{(\cdot)}(\cdot) : \{0,1\}^{l(\lambda)} \rightarrow \{0,1\}^{\ell(\lambda)}$ involves 4 algorithms ($\mathsf{ParamGen}$, KeyGen, Eval, Verify).

- $\mathsf{ParamGen}(\lambda) \rightarrow \mathsf{pp}$ takes input a security parameter λ and outputs a system public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{vk}, \mathsf{sk})$ takes input pp and outputs a verification key vk and a secret key sk.
- $\mathsf{Eval}(\mathsf{sk}, X) \rightarrow (Y, \pi)$ takes input a seed $X \in \{0,1\}^{l(\lambda)}$ and a secret key sk to generate an output $Y = F_{\mathsf{sk}}(X)$ and a proof π.
- $\mathsf{Verify}(\mathsf{vk}, X, Y, \pi) \rightarrow \{0,1\}$ checks the correctness of generated output Y with the help of the seed X, the verification key vk and the proof π.

Related Works of VRF. There are a number of key breakthroughs in the academic research of VRFs. Dodis and Yampolskiy [11] gave a compact VRF from pairing with shorter proofs and keys. Liang et al. [22] proposed a VRF from a multilinear map. Bitansky [5] and Goyal *et al.* [16] gave a generic construction of VRF from non-interactive witness-indistinguishable proof. Recently, Esgin et al. [12] proposed a lattice-based VRF with applications to the blockchain. Badertscher et al. [2] proved the UC security of EC-VRF and proposed a batch verification for EC-VRF.

Another line of work is to construct VRF with exponentially-large input spaces. Hohenberger and Waters [19] gave a VRF under a non-interactive complexity assumption. Jager [20] gave a VRF from a weaker assumption. Hofheinz and Jager [18] improved the VRF by using a non-interactive constant-size assumption. Yamada [27] proposed VRFs with either short proof or short keys. Rosie [26] constructed a VRF with a shorter key. Kohl [21] presented a VRF which supports an exponential-sized input space, achieves full adaptive security based on a non-interactive constant-size assumption and its proofs consist of only a logarithmic number of group elements for inputs of arbitrary polynomial length. Recently, Niehues [24] gave a VRF with tight security reduction to a non-interactive q-type assumption.

2.2 Assumptions

Definition 1 (Discrete logarithm (DL) assumption). *For any probabilistic polynomial time $\mathcal{A}$ and an ECC group $\mathbb{G}$ with order $p = p(\lambda)$,*

$$\Pr\left[\mathcal{A}(g, g^x) = x\right] \leq \mathsf{negl}(\lambda)\,,$$

where $x \xleftarrow{\$} \mathbb{Z}_p$.

Definition 2 (Computational Diffie-Hellman (CDH) assumption). *For any probabilistic polynomial time $\mathcal{A}$ and an ECC group $\mathbb{G}$ with order $p = p(\lambda)$,*

$$\Pr\left[\mathcal{A}(g, g^a, g^b) = g^{ab}\right] \leq \mathsf{negl}(\lambda)\,,$$

where $a, b \xleftarrow{\$} \mathbb{Z}_p$.

Definition 3 (Decisional Diffie-Hellman (DDH) assumption). *For any probabilistic polynomial time $\mathcal{A}$ and an ECC group $\mathbb{G}$ with order $p = p(\lambda)$,*

$$\left|\Pr\left[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1\right] - \Pr\left[\mathcal{A}(g, g^a, g^b, h) = 1\right]\right| \leq \mathsf{negl}(\lambda)\,,$$

where $a, b \xleftarrow{\$} \mathbb{Z}_p$ and $h \xleftarrow{\$} \mathbb{G}$.

Definition 4 (Decisional q-Diffie-Hellman inversion (q-DHI) assumption). *For any probabilistic polynomial time $\mathcal{A}$ and an ECC group $\mathbb{G}$ with order $p = p(\lambda)$,*

$$\left|\begin{array}{l} \Pr\left[\mathcal{A}(g, g^x, \ldots, g^{x^q}, g^{x^{-1}}) = 1\right] \\ -\Pr\left[\mathcal{A}(g, g^x, \ldots, g^{x^q}, h) = 1\right] \end{array}\right| \leq \mathsf{negl}(\lambda)\,,$$

where $x \xleftarrow{\$} \mathbb{Z}_p$, $h \xleftarrow{\$} \mathbb{G}$.

2.3 Zero-Knowledge Proof

A zero-knowledge argument is always a tuple of algorithms $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$. Setup takes 1^λ as input; $\mathcal{P}$ and $\mathcal{V}$ take s and t as inputs respectively; the interactive proof between $\mathcal{P}$ and $\mathcal{V}$ outputs $\langle \mathcal{P}(s), \mathcal{V}(t)\rangle = b$ where $b \in \{0, 1\}$, and we denote the interactive transcript as $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t)\rangle$. For honest $\mathcal{P}$, $s = (\mathsf{pp}, x, w)$ contains public parameter pp, ZK statement x and ZK witness w, and honest $\mathcal{V}$ takes $t = (\mathsf{pp}, x)$. The malicious $\mathcal{P}^*$ takes auxiliary input aux, which is not predefined.

Definition 5 (Interactive argument of knowledge). *An algorithm tuple $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ is called interactive argument of knowledge for a relation R when it fulfills perfect completeness and computational witness-extended emulation.*

Perfect completeness. *For any unbounded adversary $\mathcal{A}$*

$$\Pr\left[\begin{array}{l|l} (x, w) \notin R \text{ or} & \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda) \\ \langle \mathcal{P}(x, w), \mathcal{V}(x)\rangle = 1 & (x, w) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] = 1.$$

Computational witness-extended emulation. *For all deterministic polynomial time $\mathcal{P}^*$, there exists an expected polynomial time algorithm ε such that for every adversary $\mathcal{A}_1$ and $\mathcal{A}_2$,*

$$\left| \begin{array}{l} \Pr\left[\mathcal{A}_1(tr) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda) \\ (x, \mathsf{aux}) \leftarrow \mathcal{A}_2(\mathsf{pp}) \\ tr \leftarrow \langle \mathcal{P}^*(\mathsf{aux}), \mathcal{V}(\mathsf{pp}, x) \rangle \end{array} \right] - \\ \Pr\left[\begin{array}{l} \mathcal{A}_1(tr) = 1 \; and \\ tr \; is \; accepting \Rightarrow (x, w) \in R \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda) \\ (x, \mathsf{aux}) \leftarrow \mathcal{A}_2(\mathsf{pp}) \\ (tr, w) \leftarrow \varepsilon^{\mathcal{O}}(\mathsf{pp}, x) \end{array} \right] \end{array} \right| \leq \kappa,$$

where $\kappa = \kappa(\lambda)$ is the knowledge error rate and the oracle is given by $\mathcal{O} = \langle \mathcal{P}^*(\mathsf{pp}, x, \mathsf{aux}), \mathcal{V}(\mathsf{pp}, x) \rangle$. The emulation allows the rewind of the oracle with fresh randomness.

Definition 6 (Perfect honest-verifier zero-knowledge). *There exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$, the probability difference between*

$$\Pr\left[\begin{array}{l} \mathcal{A}_1(tr) = 1 \\ and \; (x, w) \in R \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda) \\ (x, w, \rho) \leftarrow \mathcal{A}_2(\mathsf{pp}) \\ tr \leftarrow \langle \mathcal{P}(\mathsf{pp}, x, w), \mathcal{V}(\mathsf{pp}, x, \rho) \rangle \end{array} \right]$$

and

$$\Pr\left[\begin{array}{l} \mathcal{A}_1(tr) = 1 \\ and \; (x, w) \in R \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda) \\ (x, w, \rho) \leftarrow \mathcal{A}_2(\mathsf{pp}) \\ tr \leftarrow \mathcal{S}(\mathsf{pp}, x, \rho) \end{array} \right]$$

is negligible where ρ is the randomness used by $\mathcal{V}$.

3 Security Models of VRF

A VRF is required to have the security properties of provability, uniqueness and pseudorandomness.

Provability. For all $\mathsf{pp} \leftarrow \mathsf{ParamGen}(1^\lambda)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, for all $X \in \{0,1\}^{l(\lambda)}$ and $(Y, \pi) \leftarrow \mathsf{Eval}(\mathsf{sk}, X)$, we have $\mathsf{Verify}(\mathsf{vk}, X, Y, \pi) = 1$.

3.1 Uniqueness

The model of *unconditional full uniqueness* means that there is no unbound adversary that can output a verification key vk, a seed X, and two output pairs (Y_0, π_0) and (Y_1, π_1) which can pass the verification, with $Y_0 \neq Y_1$.

Definition 7 (Unconditional full uniqueness). *There does not exist* $(\mathsf{vk}, X, Y_0, \pi_0, Y_1, \pi_1)$ *such that* $\mathsf{Verify}(\mathsf{vk}, X, Y_0, \pi_0) = 1$ *and* $\mathsf{Verify}(\mathsf{vk}, X, Y_1, \pi_1) = 1$ *with* $Y_0 \neq Y_1$.

```
Experiment $\mathrm{Exp}_{\mathcal{A}}^{\mathrm{Uni}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{ParamGen}(\lambda)$
$(\mathsf{vk}, X, Y_0, \pi_0, Y_1, \pi_1) \leftarrow \mathcal{A}(\mathsf{pp})$
**if** $\mathsf{Verify}(\mathsf{vk}, X, Y_0, \pi_0) = 1 \land \mathsf{Verify}(\mathsf{vk}, X, Y_1, \pi_1) = 1$
    **return** $1$
**else**
    **return** $0$.
```

Fig. 1. Computational Uniqueness.

```
Experiment $\mathrm{Exp}_{\mathcal{A}}^{\mathrm{PR},\underline{\Phi-\mathrm{RKA}}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{ParamGen}(\lambda), (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}), \mathcal{Q} \leftarrow \emptyset$
$(X^*, St) \leftarrow \mathcal{A}^{\mathsf{Eval}(\cdot \underline{,\cdot})}(\mathsf{pp}, \mathsf{vk})$
    where $\mathsf{Eval}$ on input $X$ $\underline{\text{and } \phi \in \Phi}$ :
        $\underline{\text{if } \phi(\mathsf{sk}) = \mathsf{sk}, \text{ then } \mathcal{Q} \leftarrow Q \cup \{X\}}$
        return $\mathsf{Eval}(\underline{\phi(}\mathsf{sk}\underline{)}, X)$
$(Y_0, \pi_0) \leftarrow \mathsf{Eval}(\mathsf{sk}, X^*), Y_1 \leftarrow \{0,1\}^{\ell(\lambda)}, b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathsf{Eval}(\cdot \underline{,\cdot}))}(Y_b, St),$
**if** $b' = b \land X^* \notin Q$
    **return** $1$
**else**
    **return** $0$.
```

Fig. 2. Pseudorandomness. The modifications for Φ-RKA security are in underline.

We define the model of *computational uniqueness* used by Esgin et al. [12] in Fig. 1. We define the advantage of an adversary $\mathcal{A}$ as $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Uni}}(\lambda) = \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathrm{Uni}}(\lambda) \rightarrow 1]$.

Definition 8 (Computational uniqueness). *A VRF scheme has computational uniqueness if for any PPT $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{Uni}(\lambda)$ is negligible in λ.*

We note that the model of computational uniqueness used by Ganesh et al. [13] is stronger than the one by Esgin et al. [12], in which the public parameters can also be chosen by the adversary.

Computational *trusted* uniqueness was defined similarly in [25], except that the verification key vk is honestly generated and is given to the adversary. The model in [25] did not provide any oracle access to sk.

3.2 Pseudorandomness

We define the model of *computational uniqueness* used by Esgin et al. [12] in Fig. 2. We define the advantage of an adversary $\mathcal{A}$ as $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{PR}}(\lambda) = |\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathrm{PR}}(\lambda) \rightarrow 1] - \frac{1}{2}|$.

Definition 9 (Pseudorandomness). *A VRF scheme has pseudorandomness if for any PPT $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{PR}(\lambda)$ is negligible in λ.*

3.3 RKA Security for Pseudorandomness

We define the related-key security for VRF. Since the adversary knows the secret key in the model of unconditional full uniqueness and computational uniqueness, we do not need to define RKA models for them. We note that it is possible to define the RKA security model for computational trusted uniqueness in [25], but it is out of scope for this paper.

In this paper, we define the RKA security model for pseudorandomness. Define $\varPhi$ as the class of related key derivation functions, such as additive function, multiplicative function, polynomial, etc. The adversary can ask the Eval oracle with any function $\phi \in \varPhi$. The oracle will return the output of Eval using the key $\phi(\mathsf{sk})$. The modifications of the pseudorandomness model are shown in red colour in Fig. 2. We define the advantage of an adversary $\mathcal{A}$ as $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{PR},\varPhi\text{-}\mathrm{RKA}}(\lambda) = |\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathrm{PR},\varPhi\text{-}\mathrm{RKA}}(\lambda) \rightarrow 1] - \frac{1}{2}|$.

Definition 10 ($\varPhi$-RKA Pseudorandomness) *A VRF scheme has $\varPhi$-RKA pseudorandomness if for any PPT $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{PR,\varPhi\text{-}RKA}(\lambda)$ is negligible in λ.*

In this paper, we will use the *selective* security model for the related-key attack. It means that at the beginning of the security game (before line 1 in Fig. 2), the adversary has to first output all oracle queries (X_i, ϕ_i) that he will ask. During the game, the adversary is allowed to query in any arbitrary order. The definition of selective $\varPhi$-RKA pseudorandomness can be defined similarly.

4 Building Block: Proof of Inversion Relation

As discussed in the introduction, our VRF construction requires a non-interactive zero-knowledge (NIZK) proof of an (exponent) inversion relation. We observed that the Bulletproof [6] provides a NIZK proof for inner product relation, i.e., knowing vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ such that $\boldsymbol{a} \cdot \boldsymbol{b} = c$ for some public constant c. We simplify the Bulletproof in a way that we want to prove the knowledge of γ and $1/\gamma$ (in the exponent) that $\gamma \cdot 1/\gamma = 1$.

4.1 ZK Proof of Inversion Relation

We give a (non-interactive) zero-knowledge proof of knowledge of the language:

$$\mathcal{L} = \{\gamma : \varGamma = g^{\gamma} \wedge \varTheta = h^{1/\gamma}\}.$$

The construction is given in Algorithm 1.

Algorithm 1: ZK Proof for language $\mathcal{L}$

```
Procedure Setup($\lambda$):
    pick a generator $\tilde{g}, \tilde{h} \leftarrow_s \mathbb{G}$;
    define $H_2 : \{0,1\}^* \to \mathbb{Z}_p$;
    return param $= (\tilde{g}, \tilde{h}, H_2)$;
Procedure Prove(param, $(g, h, \Gamma, \Theta), \gamma$):
    // Prove in ZK that $\log_g \Gamma \cdot \log_h \Theta = 1$
    // Commit Phase
    $\alpha, \beta \leftarrow_s \mathbb{Z}_p$;
    $S_1 = g^\alpha, S_2 = h^\beta$;
    Define polynomial $l(X) = \alpha + \gamma \cdot X$, $r(X) = \beta + \frac{1}{\gamma} \cdot X$;
    Compute polynomial $t(X) = l(X) \cdot r(X) := t_0 + t_1 X + t_2 X^2$;
    // i.e., $t_0 = \alpha\beta, t_1 = \frac{\alpha}{\gamma} + \beta \cdot \gamma, t_2 = 1$
    $\tau_0, \tau_1 \leftarrow_s \mathbb{Z}_p$;
    $T_0 = \tilde{g}^{t_0} \tilde{h}^{\tau_0}, T_1 = \tilde{g}^{t_1} \tilde{h}^{\tau_1}$;
    // Challenge Phase
    $x = H_2(g, h, \Gamma, \Theta, S_1, S_2, T_0, T_1)$;
    // Response Phase
    $z_\tau = \tau_1 x + \tau_0, z_l = l(x), z_r = r(x)$;
    return $\pi = (z_\tau, z_l, z_r, x, T_1)$;
Procedure Verify(param, $(g, h, \Gamma, \Theta), \pi$):
    Compute $T_0 = \tilde{g}^{z_l \cdot z_r - x^2} \tilde{h}^{z_\tau} T_1^{-x}$, $S_1 = g^{z_l} \Gamma^{-x}, S_2 = h^{z_r} \cdot \Theta^{-x}$;
    if $x = H_2(g, h, \Gamma, \Theta, S_1, S_2, T_0, T_1)$, return 1 ;
    else return 0;
```

Theorem 1. *Algorithm 1 fulfills computational witness-extended emulation if the DL assumption holds.*

Proof. We consider the prover and the verifier to run the protocol with a random challenge. If the transcript outputs rejection, the emulator simply outputs the transcript. Otherwise, the emulator rewinds the process until getting three different transcripts of accepting.

Observe that the following equation always holds for all accepting transcripts.

$$\tilde{g}^{z_l \cdot z_r} \tilde{h}^{z_\tau} = T_0 T_1^x \tilde{g}^{x^2}, \tag{1}$$

$$S_1 = g^{z_l} \Gamma^{-x}, \tag{2}$$

$$S_2 = h^{z_r} \Theta^{-x}. \tag{3}$$

Suppose we have three transcripts of x_i and $(z_{l,i}, z_{r,i}, z_{\tau,i})$ for $i \in [1,3]$. Find ν_i such that $\sum_{i=1}^3 \nu_i = 1$, $\sum_{i=1}^3 \nu_i x_i = 0$ and $\sum_{i=1}^3 \nu_i x_i^2 = 0$. Then we could compute

$$\alpha' = \sum_{i=1}^{3} \nu_i z_{l,i} \text{ such that } S_1 = g^{\alpha'},$$
$$\beta' = \sum_{i=1}^{3} \nu_i z_{r,i} \text{ such that } S_2 = g^{\beta'},$$
$$\tau_0' = \sum_{i=1}^{3} \nu_i z_{\tau,i}, \quad t_0' = \sum_{i=1}^{3} \nu_i z_{l,i} z_{r,i} \text{ such that } T_0 = \tilde{g}^{t_0'} \tilde{h}^{\tau_0'}.$$

Similarly, we can find μ_i such that $\sum_{i=1}^{3} \mu_i = 0$, $\sum_{i=1}^{3} \mu_i x_i = 1$ and $\sum_{i=1}^{3} \mu_i x_i^2 = 0$ and set t_1' and τ_1' as the following.

$$\tau_1' = \sum_{i=1}^{3} \mu_i z_{\tau,i}, \quad t_1' = \sum_{i=1}^{3} \mu_i z_{l,i} z_{r,i} \text{ such that } T_1 = \tilde{g}^{t_1'} \tilde{h}^{\tau_1'}.$$

Denote $\gamma' = \frac{z_{l,i} - \alpha'}{x_i}$ and $\delta' = \frac{z_{r,i} - \beta'}{x_i}$. By Eq. 2 and 3, we have:

$$\varGamma = (g^{z_{l,i}} S_1^{-1})^{1/x_i} = g^{(z_{l,i} - \alpha')/x_i} = g^{\gamma'},$$
$$\varTheta = (h^{z_{r,i}} S_2^{-1})^{1/x_i} = g^{(z_{r,i} - \beta')/x_i} = g^{\delta'}.$$

Putting the value of T_0 and T_1 into Eq. 1, we can see that if $z_{l,i} \cdot z_{r,i} \neq t_0' + t_1' x_i + x_i^2$, it breaks the discrete logarithm of $\tilde{g}$ and $\tilde{h}$. If not, we substitute the value of $z_{l,i} = \gamma' x_i + \alpha'$ and $z_{r,i} = \delta' x_i + \beta'$ into it and get:

$$(\gamma' x_i + \alpha') \cdot (\delta' x_i + \beta') = \tilde{t}_0 + \tilde{t}_1 x_i + x_i^2.$$

Since this equation is a degree two polynomial and it holds for all x_1, x_2, x_3, it implies that $\gamma' \cdot \delta' = 1$. Hence it implies that $\varGamma = g^{\gamma'}$ and $\varTheta = h^{1/\gamma'}$.

Theorem 2. *Algorithm 1 is zero-knowledge in the random oracle model.*

Proof. The simulator picks some random $z_l, z_r, z_\tau, x, y \in \mathbb{Z}_p$ and $T_1 \in \mathbb{G}$. It computes $T_0 = \tilde{g}^{z_l \cdot z_r} \tilde{h}^{z_\tau} \tilde{g}^{-x^2} T_1^{-x}$, $S_1 = g^{z_l} \varGamma^{-x}$ and $S_2 = h^{z_r} \cdot \varTheta^{-x}$. The simulator sets $x = H_2(g, h, \varGamma, \varTheta, S_1, S_2, T_0, T_1)$ in the random oracle model.

4.2 Batch Verification

Similar to the Bulletproof [6], our NIZK proof allows batch verification when multiple proofs are verified. For example, an Algorand node running the blockchain consensus algorithm may need to verify 1000 VRF proofs in parallel. Batch verification can saves the running time of computing exponentiation when the generators only depend on the public parameters. Batch verification is based on the observation that checking $g^x = 1 \wedge g^y = 1$ can be validated by selecting a random scalar α from $\mathbb{Z}_p$ and checking $g^{\alpha x+y} = 1$. With overwhelming probability, the latter equation implies that $g^x = 1 \wedge g^y = 1$ but the latter is more efficient to compute.

In order to speed up verification, we need to change the proof π to a slightly longer proof $\pi' = (z_\tau, z_l, z_r, x, T_0, T_1)$. To verify a single proof, the verifier first computes S_1, S_2 and runs $x = H_2(g, h, \Gamma, \Theta, S_1, S_2, T_0, T_1)$. Now the verifier needs to check

$$T_0 = \tilde{g}^{z_l \cdot z_r - x^2} \tilde{h}^{z_\tau} T_1^{-x}.$$

We use an index j to represent different proofs $\pi'_j = (z_{\tau,j}, z_{l,j}, z_{r,j}, x_j, T_{0,j}, T_{1,j})$. To batch verify different proofs, the verifier picks random $\alpha_j \in \mathbb{Z}_p$ and checks if

$$\prod_j T_{0,j}{}^{\alpha_j} = \tilde{g}^{\sum_j (z_{l,j} \cdot z_{r,j} - x_j^2)} \tilde{h}^{\sum_j z_{\tau,j}} \prod_j T_{1,j}^{-x_j}.$$

The total number of exponentiation is reduced from $3N$ to $2N+2$ for N proofs. This speedup comes at a price of sending an extra $T_{0,j}$ in each proof.

Finally, we also note that the verifier and the prover can use fast fixed-base exponentiation with precomputation to speed-up all the multi-exponentiations (i.e., the computation of T_0 for the verifier and the computation of T_0, T_1 for the prover).

5 Our VRF Construction

We propose a new verifiable random function (VRF). Suppose that the public key $\mathsf{vk} = g^{\mathsf{sk}}$. The verifable random function computes $F_{\mathsf{sk}}(a) = \mathsf{H}_0(a, \mathsf{H}_1(\mathsf{vk}, a)^{\frac{1}{\mathsf{sk}}})$, where $\mathsf{H}_0, \mathsf{H}_1$ are secure hash functions. The VRF is described in Algorithm 2.

5.1 Standard Security

The provability is straightforward.

Theorem 3. *Our VRF Protocol has computational uniqueness if the underlying NIZK proof has computational witness-extended emulation.*

Proof. If the adversary $\mathcal{A}$ can output $(\mathsf{vk}, X, Y_0, \pi_0, Y_1, \pi_1)$ such that $\mathcal{A}$ wins the game, we first have $\mathsf{Verify}(\mathsf{vk}, X, Y_0, \pi_0) = 1$. Denote $\pi_i = (U_i, \rho_i)$ for $i = 0, 1$. The witness-extended emulation property of the NIZK proof ρ_0 implies that there exists γ_0 such that $\mathsf{vk} = g^{\gamma_0}$ and $U_0 = \mathsf{H}_1(\mathsf{vk}, X)^{1/\gamma_0}$. Similarly for ρ_1, there exists γ_1 such that $\mathsf{vk} = g^{\gamma_1}$ and $U_1 = \mathsf{H}_1(\mathsf{vk}, X)^{1/\gamma_1}$. It implies that $\gamma_0 = \gamma_1$ and hence $U_0 = U_1$ As a result, $Y_0 = \mathsf{H}_0(X, U_0) = \mathsf{H}_0(X, U_1) = Y_1$. It reaches a contradiction that $\mathcal{A}$ wins the game with $Y_0 \neq Y_1$.

Theorem 4. *Our VRF Protocol has pseudorandomness if the DDH assumption holds in the random oracle model and the underlying NIZK proof is zero-knowledge.*

Algorithm 2: VRF Protocol

```
Procedure $\mathsf{ParamGen}(\lambda)$:
    pick a generator $g \leftarrow_s \mathbb{G}$;
    define $\mathsf{H}_0 : \{0,1\}^{l(\lambda)} \times \mathbb{G} \to \{0,1\}^{\ell(\lambda)}$;
    define $\mathsf{H}_1 : \mathbb{G} \times \{0,1\}^{l(\lambda)} \to \mathbb{G}$;
    It runs $\mathsf{param}' \leftarrow \textsc{Setup}(\lambda)$;
    return $\mathsf{pp} = (g, \mathsf{H}_0, \mathsf{H}_1, \mathsf{param}')$;
Procedure $\mathsf{KeyGen}(\mathsf{pp})$:
    $\mathsf{sk} \leftarrow_s \mathbb{Z}_p$;
    $\mathsf{vk} = g^{\mathsf{sk}}$;
    return $(\mathsf{vk}, \mathsf{sk})$;
Procedure $\mathsf{Eval}(\mathsf{sk}, X)$:
    $U = \mathsf{H}_1(\mathsf{vk}, X)^{\frac{1}{\mathsf{sk}}}$;
    $\rho \leftarrow \textsc{Prove}(\mathsf{param}', (g, \mathsf{H}_1(\mathsf{vk}, X), \mathsf{vk}, U), \mathsf{sk})$;
    $Y = \mathsf{H}_0(X, U)$;
    return $(Y, \pi = (U, \rho))$;
Procedure $\mathsf{Verify}(\mathsf{vk}, X, Y, \pi)$:
    parse $\pi = (U, \rho)$;
    if $\textsc{Verify}(\mathsf{param}', (g, \mathsf{H}_1(\mathsf{vk}, X), \mathsf{vk}, U), \rho) = 1$ and $Y = \mathsf{H}_0(X, U)$, return 1;
    else return 0;
```

Proof. Setup. The algorithm $\mathcal{B}$ is given the Divisible Decision Diffie-Hellman (DDDH) instance (g, g^a, g^b, T) and decides if $T = g^{b/a}$. Note that the DDDH problem is proved to be equivalent to the DDH problem in [3]. $\mathcal{B}$ sets $\mathsf{vk} = g^a$. $\mathcal{B}$ runs $\mathsf{param}' \leftarrow \textsc{Setup}(\lambda)$ and gives $\mathsf{pp} = (g, \mathsf{H}_0, \mathsf{H}_1, \mathsf{param}')$ and vk to the adversary $\mathcal{A}$.

Oracle Simulation.

- H_0 oracle: on input a new pair (X_i, U_i), it returns a random $Y_i \in \{0,1\}^{\ell(\lambda)}$.
- H_1 oracle: on input a new (vk, X_i), it picks a random $\gamma_i \in \mathbb{Z}_p$ and returns $(g^a)^{\gamma_i}$. Except for one time, it returns g^b.
- Eval oracle: on input X_j, it retrieves the corresponding value γ_j from the H_1 oracle query. If the output was g^b, $\mathcal{B}$ declares failure and exits. Otherwise it computes $U = g^{\gamma_j}$ and $Y = \mathsf{H}_0(X_j, U)$. $\mathcal{B}$ uses the simulator of the zero-knowledge proof to generate ρ and returns $(Y, \pi = (U, \rho))$.

Challenge. $\mathcal{A}$ returns a challenge seed X^*. $\mathcal{B}$ retrieves the corresponding value from the H_1 oracle query. If the output was not g^b, $\mathcal{B}$ declares failure and exits. Otherwise, $U^* = T$ and $Y^* = \mathsf{H}_0(X^*, U^*)$. Hence $\mathcal{B}$ returns Y^* to $\mathcal{A}$.

Output. Finally $\mathcal{A}$ outputs its guess b'. If $b' = 1$, the $\mathcal{B}$ outputs that $T = \bar{g}^{\frac{b}{a}}$. If $b' = 0$, the $\mathcal{B}$ outputs that T is randomly chosen from $\mathbb{G}$.

We can see that the success probability of the simulation is $1/q_H$, where q_H is the number of oracle queries to the H_1 oracle.

5.2 Φ^{Lin}-RKA Security

Next we show that our VRF protocol is secure against RKA attack, with respect to the class of Linear function Φ^{Lin}, where for all $\phi \in \Phi^{\mathsf{Lin}}$:

$$\phi(x) = x + C,$$

for some $C \in \mathbb{Z}_p$.

Theorem 5. *Our VRF protocol has pseudorandomness against selective Φ^{Lin}-RKA attack if the decisional q-DHI assumption holds in the random oracle model and the underlying NIZK proof is zero-knowledge, where $q-1$ is the number of oracle query to the* Eval *oracle.*

Proof. Suppose that the Eval oracle input $(\tilde{X}_i, \phi_i)$ are given to the simulator $\mathcal{B}$ are the beginning of the game, where $\phi_i(x) := x + C_i$ for $i \in [1, q-1]$. We exclude the identity function with $C_i = 0 \bmod p$ here since it is already captured by the standard Eval oracle.

Setup. The algorithm $\mathcal{B}$ is given the decisional q-DHI instance $(\bar{g}, \bar{g}^a, \ldots, \bar{g}^{a^q}, T)$ and decides if $T = \bar{g}^{1/a}$. $\mathcal{B}$ (implicitly) defines $\mathsf{sk} = a$ and defines a polynomial $P(\mathsf{sk}) = \prod_{i=1}^{q-1}(\mathsf{sk} + C_i) := \sum_{i=0}^{q-1} A_i a^i$ for some coefficients $A_i \in \mathbb{Z}_p$. $\mathcal{B}$ sets:

$$g = \bar{g}^{P(\mathsf{sk})} = \prod_{i=0}^{q-1} (\bar{g}^{a^i})^{A_i}, \quad \mathsf{vk} = g^{\mathsf{sk}} = \bar{g}^{aP(a)} = \prod_{i=1}^{q} (\bar{g}^{a^i})^{A_{i-1}}.$$

$\mathcal{B}$ runs $\mathsf{param}' \leftarrow \textsc{Setup}(\lambda)$. and gives $\mathsf{pp} = (g, \mathsf{H}_0, \mathsf{H}_1, \mathsf{param}')$ and vk to the adversary $\mathcal{A}$.

Oracle Simulation.

- H_0 oracle: on input a new pair (X_i, U_i), it returns a random $Y_i \in \{0,1\}^{\ell(\lambda)}$.
- H_1 oracle: on input a new (vk, X_i), it picks a random $\gamma_i \in \mathbb{Z}_p$ and returns $(g^a)^{\gamma_i}$. Except for one time, it picks a random $\gamma^* \in \mathbb{Z}_p$ and returns g^{γ^*}.
- Eval oracle: on input (X_i, ϕ_i), we have the following cases:
  1. ϕ_i is the identity map and $\mathsf{H}_1(\mathsf{vk}, X_i) = (g^a)^{\gamma_i}$. Then $\mathcal{B}$ sets $U = g^{\gamma_i}$ and $Y = \mathsf{H}_0(X_i, U)$. $\mathcal{B}$ uses the simulator of the zero-knowledge proof to generate ρ and returns $(Y, \pi = (U, \rho))$.
  2. $\phi_i(x) = x + C_i$ and $\mathsf{H}_1(\mathsf{vk}, X_i) = (g^a)^{\gamma_i}$. $\mathcal{B}$ computes

     $$U = g^{\frac{a\gamma_i}{\phi_i(\mathsf{sk})}} = \bar{g}^{\frac{a\gamma_i \cdot P(\mathsf{sk})}{\mathsf{sk}+C_i}}.$$

     Since $P(\mathsf{sk})$ is divisible by $\mathsf{sk} + C_i$, the value of U can be computed from the decisional q-DHI instance. $\mathcal{B}$ uses the simulator of the zero-knowledge proof to generate ρ and returns $(Y = \mathsf{H}_0(X_i, U), \pi = (U, \rho))$.
  3. ϕ_i is the identity map and $\mathsf{H}_1(\mathsf{vk}, X_i) = g^{\gamma^*}$. $\mathcal{B}$ declares failure and exits.

4. $\phi_i(x) = x + C_i$ and $\mathsf{H}_1(\mathsf{vk}, X_i) = g^{\gamma^*}$. $\mathcal{B}$ computes
$$U = g^{\frac{\gamma^*}{\phi_i(\mathsf{sk})}} = \bar{g}^{\frac{\gamma^* \cdot P(\mathsf{sk})}{\mathsf{sk}+C_i}}.$$
Since $P(\mathsf{sk})$ is divisible by $\mathsf{sk} + C_i$, the value of U can be computed from the decisional q-DHI instance. $\mathcal{B}$ uses the simulator of the zero-knowledge proof to generate ρ and returns $(Y = \mathsf{H}_0(X_i, U), \pi = (U, \rho))$.

Challenge. $\mathcal{A}$ returns a challenge seed X^*. $\mathcal{B}$ retrieves the corresponding value from the H_1 oracle query. If the hash value is not g^{γ^*}, $\mathcal{B}$ declares failure and exits. Otherwise, define coefficients $B_{-1}, B_0, B_1, ..., B_{q-2} \in \mathbb{Z}_p$ such that:

$$\frac{\gamma^* \cdot P(\mathsf{sk})}{a} = \frac{\gamma^* \cdot \prod_{i=1}^{q-1}(a + C_i)}{a} := \sum_{i=0}^{q-2} B_i a^i + \frac{B_{-1}}{a}.$$

Note that $B_{-1} \neq 0$ since $C_i \neq 0$. We have:

$$U_0 = \mathsf{H}_1(\mathsf{vk}, X^*)^{\frac{1}{\mathsf{sk}}} = \bar{g}^{\frac{\gamma^* \cdot P(\mathsf{sk})}{a}} = \prod_{i=0}^{q-2} (\bar{g}^{a^i})^{B_i} \cdot \bar{g}^{\frac{B_{-1}}{a}}.$$

Hence $\mathcal{B}$ sets $U^* = \prod_{i=0}^{q-2}(\bar{g}^{a^i})^{B_i} \cdot T^{B_{-1}}$ and returns $Y^* = \mathsf{H}_0(X^*, U^*)$ to $\mathcal{A}$.

Output. Finally $\mathcal{A}$ outputs its guess b'. If $b' = 0$, the $\mathcal{B}$ outputs that $T = \bar{g}^{\frac{1}{a}}$. If $b' = 1$, the $\mathcal{B}$ outputs that T is randomly chosen from $\mathbb{G}$.

We can see that the success probability of the simulation is at least $\frac{1}{q_H^2}(q_H - q_E)$, q_H is the number of oracle query to the H_1 oracle, and q_E is the number of oracle query to the Eval oracle.

5.3 UC Security of Unpredictability Under Malicious Key Generation

David *et. al.* [9] defined the security requirement of unpredictability under malicious key generation, which is essential for their blockchain consensus algorithm. It cannot be implied by the previous game-based definition of pseudorandomness. A UC definition of VRF capturing *unpredictability under malicious key generation* is defined by [9] and is shown in Fig. 3. Esgin et al. [12] defined a game-based version called *unbiasability*. A slightly weaker UC functionality of VRF was given in [2].

In this paper, we proved the security of our VRF under the UC security model of unpredictability in [9], because VRFs are widely used in blockchain consensus algorithms and they are mostly proved in the UC security model.

Theorem 6. *Our VRF construction realizes $\mathcal{F}_{\mathsf{VRF}}$ in the random oracle model assuming the CDH problem.*

Proof. We describe a simulator $\mathcal{S}$ that controls the random oracles H_0, H_1 and operates in the following manner.

Functionality $\mathcal{F}_{\mathsf{VRF}}$

The functionality of VRF which interacted with stakeholders U_i where $i \in [n]$, is defined as below.

- Key generation: When getting $(\mathsf{KeyGen}, \mathsf{sid})$ from U_i, transfer $(\mathsf{KeyGen}, \mathsf{sid}, U_i)$ to the adversary. When receiving $(\mathsf{VerificationKey}, \mathsf{sid}, U_i, \mathsf{vk})$ from the adversary, with honest U_i and unique vk, save the tuple (U_i, vk), initialize $T(\mathsf{vk}, \cdot) = \emptyset$, and return $(\mathsf{VerificationKey}, \mathsf{sid}, \mathsf{vk})$ to U_i.
- VRF evaluation: When getting $(\mathsf{Eval}, \mathsf{sid}, X)$ from U_i, ignore it if (U_i, vk) is not recorded before. If $T(\mathsf{vk}, X)$ has not been set, set a random $T(\mathsf{vk}, X) \leftarrow (Y, \emptyset)$ for a random $Y \xleftarrow{\$} \{0,1\}^{\ell}$. Return $(\mathsf{Evaluated}, \mathsf{sid}, Y)$ to a party P, where $T(\mathsf{vk}, X) = (Y, \emptyset)$.
- VRF evaluation and proof: When getting $(\mathsf{EvalProof}, \mathsf{sid}, X)$ from U_i, ignore the request if (U_i, vk) is not recorded before. Else, send $(\mathsf{EvalProof}, \mathsf{sid}, U_i, X)$ to the adversary. When getting $(\mathsf{Evaluated}, \mathsf{sid}, X, \pi)$ from the adversary, set $T(\mathsf{vk}, X) \leftarrow (Y, \emptyset)$ for a random $Y \xleftarrow{\$} \{0,1\}^{\ell}$ if $T(\mathsf{vk}, X)$ is not found. Else if $T(\mathsf{vk}, X) = (Y, S)$ for some set S, set $T(\mathsf{vk}, X) \leftarrow (Y, S \cup \{\pi\})$.
- Verification: When getting message $(\mathsf{Verify}, \mathsf{sid}, X, Y, \pi, \mathsf{vk}')$ from some party P, forward it to the adversary. When receiving $(\mathsf{Verified}, \mathsf{sid}, X, Y, \pi, \mathsf{vk}')$ from the adversary, check its integrity. If (U_i, vk') is not stored, initialize $T(\mathsf{vk}', \cdot) = \emptyset$ and set $f = 0$; otherwise, set $f = 1$ if there exists $\pi \in S$ where $T(U_i, X) = (Y, S)$; set $f = 0$ on all other cases. Return $(\mathsf{Verified}, \mathsf{sid}, X, Y, \pi, f)$ to P.
- Malicious key generation: When getting $(\mathsf{KeyGen}, \mathsf{sid}, \mathsf{vk})$ from a malicious party $\mathcal{S}$, save $(\mathcal{S}, \mathsf{vk})$ and initialize $T(\mathsf{vk}, \cdot) = \emptyset$ if vk is not recorded before.
- Malicious VRF evaluation: When getting a message $(\mathsf{Eval}, \mathsf{sid}, \mathsf{vk}, X)$ from $\mathcal{S}$, if $T(\mathsf{vk}, X) = (Y, S)$ for some $S \neq \emptyset$ and $(\mathcal{S}, \mathsf{vk})$ have been defined, return $(\mathsf{Evaluated}, \mathsf{sid}, Y)$ to $\mathcal{S}$. If $T(\mathsf{vk}, X)$ has not been defined, set $T(\mathsf{vk}, X) \leftarrow (Y, \emptyset)$ for $Y \xleftarrow{\$} \{0,1\}^{\ell}$ and return $(\mathsf{Evaluated}, \mathsf{sid}, Y)$ to $\mathcal{S}$. Ignore this request in all other cases.

Fig. 3. Functionality $\mathcal{F}_{\mathsf{VRF}}$

1. Upon receiving a message $(\mathsf{KeyGen}, \mathsf{sid}, U_i)$ from $\mathcal{F}_{\mathsf{VRF}}$, a new value $\mathsf{vk} = g^k$ is selected for a random k and $\mathcal{S}$ inserts (U_i, vk) in its internal registry of keys; in case the key exists already, $\mathcal{S}$ returns fail and terminate. It returns to $\mathcal{F}_{\mathsf{VRF}}$ the message $(\mathsf{VerificationKey}, \mathsf{sid}, U_i, \mathsf{vk})$.
2. Upon receiving a message $(\mathsf{EvalProof}, \mathsf{sid}, U_i, X)$ from $\mathcal{F}_{\mathsf{VRF}}$, this is matched to the verification key vk of U_i and is checked whether X has been queried before. In such a case, the value U that corresponds to X in the table for vk is recovered. In case X was not queried before, it is checked whether $\mathsf{H}_1(\mathsf{vk}, X)$ is defined. In such case the entry $(\mathsf{base}, \mathsf{vk}, X, \gamma, h_1)$ is recovered, the value U is set to g^{γ} and the triple (vk, X, U) is stored for future reference. In case the value $\mathsf{H}_1(\mathsf{vk}, X)$ is undefined $\mathcal{S}$ selects γ at random from $\mathbb{Z}_p$, stores $(\mathsf{base}, \mathsf{vk}, X,\ \gamma, h_1 = \mathsf{vk}^{\gamma})$ and sets $\mathsf{H}_1(\mathsf{vk}, X) = h_1$.

   Subsequently random values $x, z_{\tau}, z_l, z_r \in \mathbb{Z}_p$ and $T_1 \in \mathbb{G}$ are selected by $\mathcal{S}$. $\mathcal{S}$ computes $T_0 = \tilde{g}^{z_l \cdot z_r} \tilde{h}^{z_{\tau}} \tilde{g}^{-x^2} T_1^{-x}$, $S_1 = g^{z_l} \mathsf{vk}^{-x}, S_2 = \mathsf{H}_1(\mathsf{vk}, X)^{z_r} \cdot U^{-x}$. The pair $((g, \mathsf{H}_1(v, X)), \mathsf{vk}, U,\ S_1, S_2, T_0, T_1), x)$ is inserted to the table of the

Table 1. Security of practical VRF schemes and their assumptions. RO stands for the random oracle model. × means insecure.

	Uniqueness	Pseudorandomness	UC Security of Unpredictability	RKA Pseudorandomness
EC-VRF [25]	RO	DDH + RO	(CDH + RO using model in [2])	×
[9]			CDH + RO	×
This paper	DL + RO	DDH + RO	CDH + RO	q-DHI + RO

random oracle H_0. In case this is not feasible (because that would make the table inconsistent), $\mathcal{S}$ outputs fail and terminates. Finally, π is set to $(U, \rho = (z_\tau, z_l, z_r, x, T_1))$ and the message $(\mathsf{Eval}, \mathsf{sid}, X, \pi)$ is returned to $\mathcal{F}_{\mathsf{VRF}}$.

3. Upon receiving $(\mathsf{Verify}, \mathsf{sid}, X, Y, \pi, \mathsf{vk}')$ from $\mathcal{F}_{\mathsf{VRF}}$, parse π as (U', ρ') and verify the proof ρ' as a proof of $\log_g(\mathsf{vk}') = 1/\log_{\mathsf{H}_1(\mathsf{vk},X)}(U')$, to obtain a bit b. Now observe that $(\mathsf{base}, \mathsf{vk}', X, \gamma, h_1)$ must be recorded, and set $b' = ((U')^{1/\gamma} = g) \wedge (\mathsf{H}_0(X, U) = Y)$. If $b \neq b'$ output fail and terminate. In any other case, return $(\mathsf{Verified}, \mathsf{sid}, X, Y, \pi, \mathsf{vk}')$ to $\mathcal{F}_{\mathsf{VRF}}$.
4. Upon receiving a query (vk, X) for the random oracle H_1, select $\gamma \in \mathbb{Z}_p$ at random, store $(\mathsf{base}, \mathsf{vk}, X, \gamma, h_1 = \mathsf{vk}^\gamma)$ and return h_1.
5. Upon receiving a query for the random oracle H_0 of the form (X, U), and the value $(\mathsf{base}, \mathsf{vk}, X, \gamma, h_1)$ is stored previously, $\mathcal{S}$ performs the following. First, if (g, U, vk, h_1) is a CDH tuple (via checking if $U = g^\gamma$) and vk is not registered as a public-key it registers $(\mathsf{KeyGen}, \mathsf{sid}, \mathsf{vk})$ with $\mathcal{F}_{\mathsf{VRF}}$. Then it submits $(\mathsf{Eval}, \mathsf{sid}, \mathsf{vk}, m)$ to the $\mathcal{F}_{\mathsf{VRF}}$. If $\mathcal{F}_{\mathsf{VRF}}$ ignores the request $\mathcal{S}$ terminates with fail. Else it obtains the response Y which is set as the random oracle output to the query (X, U). In case $(\mathsf{base}, \mathsf{vk}, X, \gamma, h_1)$ is not stored, then perform the step that corresponds to the query $\mathsf{H}_1(\mathsf{vk}, X)$ above and repeats the current step. Other queries to H_0 are handled by returning random elements of $\{0,1\}^{\ell(\lambda)}$.

We observe that unless $\mathcal{S}$ outputs fail, the simulation is perfect. We next argue that the probability of output failure is negligible. $\mathcal{S}$ outputs fail in the case that $\mathcal{F}_{\mathsf{VRF}}$ ignores a request $(\mathsf{Eval}, \mathsf{sid}, \mathsf{vk}, X)$. This means that the key vk is registered with an honest party that has not evaluated X. It follows that the event an adversary that produces such U can be turned into a solver for the Diffie-Hellman Inversion (DHI) problem. Note that the DHI problem is proved to be equivalent to the CDH problem in [3]. The other cases where $\mathcal{S}$ produces fail, specifically, step 1, 2 and 3 can be easily seen to be negligible probability events.

Then, we can build an attacker against the DHI assumption as follows. Let (g, g^a) define an instance of the DHI problem. We run the above simulator $\mathcal{S}$ against the given adversary by choosing one of the honest users at random and setting its $\mathsf{vk} = g^a$.

For H_1 query with $(\mathsf{vk} = g^a, X)$, select $\gamma \in \mathbb{Z}_p$ at random, store $(\mathsf{base}, \mathsf{vk} = g^a, X, \gamma, h_1 = \mathsf{vk}^\gamma)$ and return h_1. Except for one time with input $(\mathsf{vk} = g^a, X)$, select $\gamma^* \in \mathbb{Z}_p$ at random, store $(\mathsf{base}, \mathsf{vk} = g^a, X, \gamma^*, h_1^* = g^{\gamma^*})$ and return h_1^*.

If $\mathcal{S}$ fails in step 5, it means that $\mathcal{S}$ receives a query (X, U). If $\mathsf{H}_1(\mathsf{vk}^*, X) = h_1^*$ (with probability at least $1/q_{\mathsf{H}_1}$, where q_{H_1} is the number of oracle queries to H_1), $\mathcal{S}$ fails when $(g, U, g^a, g^{\gamma^*})$ is a CDH tuple, i.e., $U = g^{\gamma^*/a}$. Then $\mathcal{S}$ returns U^{1/γ^*} as the solution to the DHI problem.

6 Analysis and Implementation

Security. We first compare the security of the practical VRF schemes EC-VRF [25], the VRF in [9] and our scheme in Table 1. We note that the construction of [9] is very similar to EC-VRF (applying another hash function over the EC-VRF output with the seed X). Hence, the uniqueness and pseudorandomness of [9] should be similar to that of EC-VRF.

For the recent lattice-based VRF [12], they achieve pseudorandomness (under the MLWE assumption, in the random oracle model), computational uniqueless (under the MSIS assumption, in the random oracle model), and unbiasability (the game-based version of unpredicatability) in the random oracle model. Their lattice-based VRF has the limitation of generating a limited number k of VRF outputs only for a single key pair. In their concrete parameter setting, they set $k = 1, 3, 5$ only. Hence, we do not include their scheme for direct comparison.

Efficiency. The running time of EC-VRF and the VRF in [9] are very similar due to the similarity of their constructions. Hence, we only compare EC-VRF with our scheme in the efficiency analysis and the implementation part. In EC-VRF, the running time of Eval is dominated by two exponentiations in $\mathbb{G}$. The running time of Verify is dominated by four exponentiations in $\mathbb{G}$.

In our VRF, the running time of Eval is dominated by seven exponentiations in $\mathbb{G}$. The running time of Verify is dominated by seven exponentiations in $\mathbb{G}$. The running time for verification is more important for blockchain consensus algorithms. We did not implement the batch verification in our implementation. As discussed in Sect. 4.2, our NIZK proof allows batch verification when multiple VRFs are verified. In [6], a single verification takes 3.9ms, while a batch verification of 64 proofs takes 1.9ms per proof. The running time of Verify can be further sped up by applying such a batch verification technique.

Implementation. We implement EC-VRF and our scheme in a virtual machine with Ryzen 5800H CPU @ 3.2 GHz, and 4 GB memory. They are both written in Rust using Curve25519. The source code of our implementation has been updated to GitHub[1]. After 1000 repetitions, the average running time for Eval and Verify time of EC-VRF are 0.17 ms and 0.22 ms respectively. The average running time for Eval and Verify time of our scheme are 1.025 ms and 1.049 ms respectively. Our scheme is slower because of the larger number of exponentiation, the computation of modular inverse, and two more hash function computations. We note that our scheme is still very efficient with a running time of about 1 ms, offers a higher level of security, and can be further optimized by using batch verification. Algorand's mainnet currently employs over 1000 nodes

[1] https://github.com/rka-vrf/rka-vrf.

which runs a VRF protocol for the consensus algorithms. The verification time for 1000 VRF takes around 1 s even if we do not use batch verification. It is less than the block generation time of 5 s in Algorand.

7 Conclusion

In this paper, we present a new VRF scheme that is secure against the standard definition of uniqueness, pseudorandomness, and UC security of unpredictability under malicious key generation. We define the *first* security model of RKA security against pseudorandomness for VRF. We show that our construction is also secure under this strong model. Our scheme is practical and can be used in real-world applications like the blockchain.

References

1. Abdalla, M., Benhamouda, F., Passelègue, A., Paterson, K.G.: Related-key security for pseudorandom functions beyond the linear barrier. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 77–94. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_5
2. Badertscher, C., Gaži, P., Querejeta-Azurmendi, I., Russell, A.: On uc-secure range extension and batch verification for ecvrf. Cryptology ePrint Archive, Paper 2022/1045 (2022). https://eprint.iacr.org/2022/1045. To appear in ESORICS 2022
3. Bao, F., Deng, R.H., Zhu, H.F.: Variations of diffie-hellman problem. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 301–312. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39927-8_28
4. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_31
5. Bitansky, N.: Verifiable random functions from non-interactive witness-indistinguishable proofs. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 567–594. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_19
6. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: IEEE SP 2018, pp. 315–334. IEEE Computer Society (2018)
7. Burdges, J., et al.: Overview of polkadot and its design considerations. Cryptology ePrint Archive, Paper 2020/641 (2020). https://eprint.iacr.org/2020/641
8. Chow, S.S.M., Hui, L.C.K., Yiu, S.M., Chow, K.P.: An e-Lottery scheme using verifiable random function. In: Gervasi, O., et al. (eds.) ICCSA 2005. LNCS, vol. 3482, pp. 651–660. Springer, Heidelberg (2005). https://doi.org/10.1007/11424857_72
9. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: an adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 66–98. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_3

10. Docs, C.: Introduction to chainlink vrf (2020). https://docs.chain.link/docs/chainlink-vrf/#overview
11. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30580-4_28
12. Esgin, M.F., et al.: Practical post-quantum few-time verifiable random function with applications to algorand. In: Borisov, N., Diaz, C. (eds.) FC 2021. LNCS, vol. 12675, pp. 560–578. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64331-0_29
13. Ganesh, C., Orlandi, C., Tschudi, D.: Proof-of-stake protocols for privacy-aware blockchains. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 690–719. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_23
14. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling byzantine agreements for cryptocurrencies. In: SOSP 2017, pp. 51–68. ACM (2017)
15. Goldberg, S., Reyzin, L., Papadopoulos, D., Včelák, J.: Verifiable random functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf-15, Internet Engineering Task Force (2022). https://datatracker.ietf.org/doc/draft-irtf-cfrg-vrf/15/. work in Progress
16. Goyal, R., Hohenberger, S., Koppula, V., Waters, B.: A generic approach to constructing and proving verifiable random functions. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 537–566. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_18
17. Hanke, T., Movahedi, M., Williams, D.: DFINITY technology overview series, consensus system. CoRR abs/1805.04548 (2018). http://arxiv.org/abs/1805.04548
18. Hofheinz, D., Jager, T.: Verifiable random functions from standard assumptions. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 336–362. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_14
19. Hohenberger, S., Waters, B.: Constructing verifiable random functions with large input spaces. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 656–672. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_33
20. Jager, T.: Verifiable random functions from weaker assumptions. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 121–143. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_5
21. Kohl, L.: Hunting and gathering – verifiable random functions from standard assumptions with short proofs. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11443, pp. 408–437. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17259-6_14
22. Liang, B., Li, H., Chang, J.: Verifiable random functions from (Leveled) multilinear maps. In: Reiter, M., Naccache, D. (eds.) CANS 2015. LNCS, vol. 9476, pp. 129–143. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26823-1_10
23. Micali, S., Reyzin, L.: Soundness in the public-key model. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 542–565. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_32
24. Niehues, D.: Verifiable random functions with optimal tightness. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12711, pp. 61–91. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_3
25. Papadopoulos, D., et al.: Making nsec5 practical for dnssec. Cryptology ePrint Archive, Paper 2017/099 (2017). https://eprint.iacr.org/2017/099

26. Roşie, R.: Adaptive-secure VRFs with shorter keys from static assumptions. In: Camenisch, J., Papadimitratos, P. (eds.) CANS 2018. LNCS, vol. 11124, pp. 440–459. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00434-7_22
27. Yamada, S.: Asymptotically compact adaptively secure lattice IBEs and verifiable random functions via generalized partitioning techniques. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 161–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_6

Statistically Consistent Broadcast Authenticated Encryption with Keyword Search

Adaptive Security from Standard Assumptions

Sayantan Mukherjee(✉)

Department of Computer Science and Engineering,
Indian Institute of Technology, Jammu, India
csayantan.mukherjee@gmail.com

Abstract. Searchable Encryption (SE) allows users to perform a keyword search over encrypted documents. In Eurocrypt'04, Boneh et al. introduced Public-key Encryption with Keyword Search (PEKS). Broadcast Encryption with Keyword Search (BEKS) is a natural progression to allow some amount of access control. Unfortunately, PEKS and BEKS suffer from keyword-guessing attacks (KGA). In the case of KGA, an adversary guesses the keyword encoded in a trapdoor by creating a ciphertext on a sequence of keywords of its choice and testing them against the trapdoor. In ACISP'21, Liu et al. introduced a variant of BEKS called Broadcast Authenticated Encryption with Keyword Search (BAEKS), which tried to mitigate KGA in BEKS. This construction did not argue consistency and achieved weaker security in the random oracle model.

In this work, we first introduce the notion of consistency for BAEKS and introduce security models much stronger than those of Liu et al. We propose a new statistically-consistent construction of BAEKS in the standard model that achieves security in the newly introduced models. Our proposal is proven adaptively secure under the well-studied bilateral Matrix Diffie-Hellman Assumption and still achieves asymptotic efficiency similar to that of Liu et al.

1 Introduction

With the advent of new technologies, third-party cloud storage, and network bandwidth are becoming more accessible. One now can store the data in the cloud and download them as and when required. All user in this setting puts unsubstantiated trust on the third-party server. A realistic user might encrypt the data to achieve access control. Broadcast encryption (BE) [5] has already been found to be useful to restrict the unprivileges. However, BE does not support searching on encrypted data.

Boneh *et al.* [6] kickstarted the study of public key encryption with keyword search (PEKS). It was followed by seminal works such as [1,8]. Being public key

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 523–552, 2023.
https://doi.org/10.1007/978-3-031-35486-1_23

encryptions, all of these schemes were vulnerable to keyword guessing attacks [9,23]. Two possible directions to thwart key guessing attack were explored:

- Restrict searching capability: This line of work studied designated-tester PEKS [3,14,16]. Here, Test uses the tester's secret key.
- Restrict encryption capability: This line of work studied public key authenticated encryption with keyword search (PAEKS) [12,18,21]. Here, Encrypt uses the sender's secret key.

Among the two, recently introduced PAEKS seems to be the more realistic and is therefore getting much attention of late. PAEKS allows a sender to encrypt a keyword for a particular receiver. Supporting several users to perform a keyword search in the encrypted domain has been studied for a long time [2,7]. All the existing works have tried to combine broadcast encryption (BE) with PEKS. Unfortunately, neither of these works withstand the dreaded key guessing attack. In ACISP 2021, Liu et al. [20] extended the PAEKS primitive towards supporting multiple receivers. In particular, they introduced the notion of *broadcast authenticated encryption with keyword search* (BAEKS) to allow a sender to encrypt a keyword for multiple receivers. They also have proposed security games to model (some) real-world vulnerabilities. Despite being the trendsetter, [20] suffers multiple shortcomings. To name a few, [20] was only proven non-adaptively secure in the random oracle model where the challenger would choose the challenge receiver/sender. Moreover, they did not consider sender privacy, and no formal arguments for anonymity and trapdoor anonymity were provided. On top of that, [20] does not comment about its consistency [1] despite being a searchable encryption.

1.1 Our Contribution

Liu et al. [20] introduced the notion of *broadcast authenticated encryption with keyword search* and proposed security models. They introduced four security models two for trapdoor security and two for ciphertext security. We first observe some unnatural restrictions in all of their security models. We further note that they did neither consider the security of the *sender information*, which is usually another essential anonymity requirement, nor did they provide any argument on the *consistency* of their construction.

Thus, We propose a novel definition of consistency for the BAEKS primitive, which improved upon the consistency definition of PAEKS [13]. We also propose two security models, one to capture *trapdoor security* and one for *ciphertext security*, where both consider hiding the keyword, the sender information, and the receiver(s) information simultaneously. Our security models allow user public key queries not allowed in [20]. We believe, these new security definitions would serve as a good reference for BAEKS which in a way generalizes PAEKS. Then we give a new *statistically-consistent* construction of *broadcast authenticated encryption with keyword search* BaEKS. While proving statistical consistency, we introduce a novel technique over the existing techniques [1,13]. Our construction achieves adaptive ciphertext security and trapdoor security in the standard model under standard assumptions.

1.2 Technical Overview

We start from the BEKS construction of [10] even though there are some glaring differences between their achievements and our requirements. [10] encoded a keyword and receiver set pair $(\omega, \mathcal{R})$ via a merge of Boneh-Boyen Hash (BB-Hash) [4] and Gentry-Waters hash [17]. In particular, $(\omega, \mathcal{R})$ pair was encoded as $(w_{n+1}\omega + \sum_{i\in\mathcal{R}} w_i)$ where $w_1, \ldots, w_n$ captures users and w_{n+1} is used to compute the BB-Hash [4]. To encode the (ω, R) pair in the trapdoor, [10] has to encode all w_1 to w_n one by one. They needed $\mathcal{R}$ to be given in plain for the decryption. To summarise, [10] (i) supports only polynomial many users whose keys are generated by a central authority, (ii) does not provide trapdoor privacy, (iii) does not hide receiver information, (iv) does not consider sender anonymity.

We start by encoding the *sender information* S by w_{n+1}; this results in a hash that encodes the triple $(\mathsf{S}, \omega, \mathcal{R})$. However, this encoding hides ω and S from an adversary but still leaks $\mathcal{R}$. To address this, we split this encoding to $|\mathcal{R}|$ many BB-Hash [4]arriving at $\{(w_{\mathsf{S}}\omega + w_{\mathsf{R}})\}_{\mathsf{R}\in\mathcal{R}}$ to encode $(\mathsf{S}, \omega, \mathcal{R})$. This encoding not only hides $\mathcal{R}$ but also allows each user U to choose their own w_{U} removing the necessity of a central authority.

To encode a triplet of the sender, keyword, and receiver information $(\widetilde{\mathsf{S}}, \tilde{\omega}, \mathsf{R})$ in a trapdoor, we remove some parts from the encoding of [10]. Precisely, we keep only the BB-Hash of $(\widetilde{\mathsf{S}}, \tilde{\omega}, \mathsf{R})$ i.e. $(w_{\widetilde{\mathsf{S}}}\tilde{\omega} + w_{\mathsf{R}})$. Informally speaking, the BB-Hash $(w_{\widetilde{\mathsf{S}}}\tilde{\omega} + w_{\mathsf{R}})$ in trapdoor allows us to test the *match* with BB-Hash $(w_{\mathsf{S}}\omega + w_{\mathsf{R}_i})$ in ciphertext quite naturally and also hides all of $(\widetilde{\mathsf{S}}, \tilde{\omega}, \mathsf{R})$.

We now focus on instantiation. [10] did not provide trapdoor privacy and, therefore, could afford to use the asymmetric structure of Jutla-Roy [19]. We follow Chen *et al.* [11] to make the ciphertexts and the trapdoors.

1.3 Organization of the Paper

In Sect. 2, we discuss some basic notations and descriptions of mathematical tools. We define *broadcast authenticated encryption with keyword search* and its security in Sect. 3. Then in Sect. 4, we propose a new construction BaEKS along with a security proof and present a comparison with the state of the art in Sect. 5. Finally, we conclude the paper in Sect. 6. We revisit the security models of [20] in Appendix A.

2 Mathematical Tools and Preliminaries

Notations. For $a, b \in \mathbb{N}$ such that $a \leq b$, we often use $[a, b]$ to denote $\{a, \ldots, b\}$ and write $[b] := [1, b]$. For a set D, we write $x \hookleftarrow D$ to denote x is sampled uniformly at random from D. The abbreviation ppt stands for probabilistic polynomial time. The small-case bold letters $\mathbf{s}$ denote a column vector, and large-case bold letters $\mathbf{S} = \left(s_{ij}\right)_{(i,j)\in I\times J}$ to denote a matrix of appropriate size.

2.1 Mathematical Tools and Hardness Assumptions

2.1.1 Bilinear Pairing

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}$ are groups of order p which is a large prime number. We consider an efficiently computable and non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_{\mathrm{T}}$ where $\forall g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for any $a, b \in \mathbb{Z}_p$. We use type-3 bilinear pairing in this work where $\mathbb{G}_1$ and $\mathbb{G}_2$ have no known isomorphism. We define ABSGen as the group generator that generates the type-3 bilinear pairing description. Precisely, $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, g_1, g_2) \hookleftarrow \mathsf{ABSGen}(1^\lambda)$ such that $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}$ are cyclic groups of prime order p where $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$. We denote g_s^a by $[\![a]\!]_s$ [15] for any $a \in \mathbb{Z}_p$ and $s \in \{1, 2, \mathrm{T}\}$. For $\mathbf{A} = (a_{ij}) \in \mathbb{Z}_p^{\beta \times \alpha}$, for $s \in \{1, 2\}$ we denote

$$[\![\mathbf{A}]\!]_s = \begin{pmatrix} g_s^{a_{11}} & \dots & g_s^{a_{1\alpha}} \\ \vdots & \ddots & \vdots \\ g_s^{a_{\beta 1}} & \dots & g_s^{a_{\beta\alpha}} \end{pmatrix} \in \mathbb{G}_s^{\beta \times \alpha}.$$

Definition 1. *Let $m, k \in \mathbb{N}$, such that $m > k$. We call $\mathcal{D}_{m,k}$ a matrix distribution if it outputs a matrix $\boldsymbol{M} \in \mathbb{Z}_p^{m \times k}$ of full rank k in polynomial time (w.l.o.g. we assume the first k rows of $\boldsymbol{M} \leftarrow \mathcal{D}_{m,k}$ form an invertible matrix). We write $\mathcal{D}_k = \mathcal{D}_{k+1,k}$.*

2.1.2 Bilateral Matrix Diffie-Hellman Assumption

For all adversary $\mathcal{A}$, its advantage against the bilateral matrix Diffie-Hellman problem is defined as $\mathsf{Adv}_{\mathcal{A},\mathcal{G}}^{bil\text{-}\mathcal{D}_{\ell,k}\text{-}\mathsf{matDH}}(\lambda) =$

$$|\Pr[\mathcal{A}([\![\mathbf{U}]\!]_1, [\![\mathbf{Ux}]\!]_1, [\![\mathbf{U}]\!]_2, [\![\mathbf{Ux}]\!]_2) = 1] - \Pr[\mathcal{A}([\![\mathbf{U}]\!]_1, [\![\mathbf{z}]\!]_1, [\![\mathbf{U}]\!]_2, [\![\mathbf{z}]\!]_2) = 1]|$$

where $\mathbf{U} \leftarrow \mathcal{D}_{\ell,k}$, $\mathbf{x} \hookleftarrow \mathbb{Z}_p^k$ and $\mathbf{z} \hookleftarrow \mathbb{Z}_p^\ell$. The *bil-$\mathcal{D}_{\ell,k}$*-$\mathsf{matDH}$ assumption states that $\mathsf{Adv}_{\mathcal{A},\mathcal{G}}^{bil\text{-}\mathcal{D}_{\ell,k}\text{-}\mathsf{matDH}}(\lambda)$ is negligible in λ for all ppt adversary $\mathcal{A}$.

3 Broadcast Authenticated Encryption with Keyword Search

A BAEKS scheme BaEKS is defined by a tuple of five ppt algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SrchEnc}, \mathsf{Trapdoor}, \mathsf{Test}$) as following.

3.1 Definition

- $\mathsf{Setup}(1^\lambda)$: It takes the security parameter 1^λ and publishes params pp.
- $\mathsf{KeyGen}(\mathsf{pp})$: It takes the public parameter pp as input, and outputs public-private key pair $(\mathsf{pk}, \mathsf{sk})$. It keeps sk secret and publishes pk.
- $\mathsf{SrchEnc}(\mathsf{pp}, \mathsf{sk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$: It takes pp, a sender secret key sk_{S}, a keyword ω and public-keys $\mathsf{pk}_{\mathcal{R}} = \{\mathsf{pk}_{\mathsf{R}_1}, \dots, \mathsf{pk}_{\mathsf{R}_\ell}\}$ of a set of receivers $\mathcal{R} = \{\mathsf{R}_1, \dots, \mathsf{R}_\ell\}$ where $\ell = \mathsf{poly}(\lambda)$. It outputs a ciphertext $\mathsf{Ct} \in \mathcal{CT}$.
- $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\tilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{sk}_{\mathsf{R}})$: It takes pp, a sender public key $\mathsf{pk}_{\tilde{\mathsf{S}}}$, a keyword $\tilde{\omega}$ and a receiver secret key sk_{R}. It outputs a trapdoor $\mathsf{Tr} \in \mathcal{T}$.
- $\mathsf{Test}(\mathsf{Tr}, \mathsf{Ct})$: It takes Tr and Ct as input, and outputs $0/1$.

Correctness. For any security parameter 1^λ, any sender $\mathsf{S} \in \mathcal{U}$, any receiver set $\mathcal{R} \subset \mathcal{U}$ such that $\mathsf{pk}_{\mathcal{R}} = (\mathsf{pk}_{\mathsf{U}})_{\mathsf{U} \in \mathcal{R}}$, any keyword $\omega \in \mathcal{KW}$ such that if $\mathsf{R} \in \mathcal{R}$,

$$\Pr[\mathsf{Test}(\mathsf{Trapdoor}(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{sk}_{\mathsf{R}}), \mathsf{SrchEnc}(\mathsf{sk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})) = 1] = 1 - \mathsf{neg}(\lambda)$$

where the probability is taken over $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\mathsf{sk}_{\mathsf{U}}, \mathsf{pk}_{\mathsf{U}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for all the users $\mathsf{U} \in \mathcal{R} \sqcup \{\mathsf{S}\}$.

Informally speaking, if $(\widetilde{\mathsf{S}} = \mathsf{S}) \wedge (\tilde{\omega} = \omega) \wedge (\mathsf{R} \in \mathcal{R})$, we say the key-attributes $(\widetilde{\mathsf{S}}, \tilde{\omega}, \mathsf{R})$ *match* the ciphertext-attributes $(\mathsf{S}, \omega, \mathcal{R})$. If $\mathsf{Test}(\mathsf{Tr}, \mathsf{Ct}) \rightarrow 1$, we say the trapdoor Tr *matches* the ciphertext Ct.

Consistency. We give a new definition of consistency that extends the consistency definition of PAEKS [13]. Let $\mathsf{BaEKS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SrchEnc}, \mathsf{Trapdoor}, \mathsf{Test})$ be a BAEKS scheme. We define the following experiment:

```
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$ for $i \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{R}_0, \mathsf{R}_1\}$
$(\omega_0, \omega_1, i, j) \leftarrow \mathcal{A}\left((\mathsf{pk}_t)_{t \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{R}_0, \mathsf{R}_1\}}\right)$
  s.t. $\omega_0, \omega_1 \in \mathcal{KW} \;\wedge$ distinct $i, j \in \{0,1\} \wedge ((\mathsf{S}_i, \omega_i, \mathsf{R}_i)$ *does not match* $(\mathsf{S}_j, \omega_j, \mathcal{R}_j))$
$\mathsf{Ct} \leftarrow \mathsf{SrchEnc}(\mathsf{sk}_{\mathsf{S}_i}, \omega_i, \mathsf{pk}_{\mathcal{R}_i})$
$\mathsf{Tr} \leftarrow \mathsf{Trapdoor}(\mathsf{pk}_{\mathsf{S}_j}, \omega_j, \mathsf{sk}_{\mathsf{R}_j})$
If $\mathsf{Test}(\mathsf{Tr}, \mathsf{Ct}) = 1$, Output 1 or 0 Otherwise.
```

3.2 System Model

We assume a trusted third party generates the system params pp. This is a one-time procedure. Following this, any user U can generate its own public-private key pairs $(\mathsf{pk}_{\mathsf{U}}, \mathsf{sk}_{\mathsf{U}})$. To compute a searchable ciphertext of a keyword ω for a set of users (called receiver-set $\mathcal{R} = \{\mathsf{R}_1, \ldots, \mathsf{R}_\ell\}$), a user (called sender S) first collects the receivers' public keys $\mathsf{pk}_{\mathcal{R}} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$. S then encrypts the keyword ω to Ct running $\mathsf{SrchEnc}$ on its own secret key sk_{S}. Any user can compute the trapdoor Tr for a keyword $\tilde{\omega}$ of interest with its own secret key sk_{R} and a sender's public key $\mathsf{pk}_{\widetilde{\mathsf{S}}}$, and send Tr to the cloud server for a search query. The cloud server can test on Ct and Tr without knowing neither the sender's identity nor the receiver's identity. A document will be returned if all the followings hold: the underlying keywords are the same $(\omega = \tilde{\omega})$, the trapdoor Tr is for querying the content from the sender S (i.e. $\mathsf{S} = \widetilde{\mathsf{S}}$) rather than other senders, and the receiver R is one of the target receivers (i.e. $\mathsf{R} \in \mathcal{R}$) of the searchable ciphertext Ct. On a glance, this system model will look quite different from that of [20] as we do not consider KGC running KeyGen for all users unlike in [20]. A careful look at [20] shows that in a BAEKS, KGC does not create/hold any master secret key and therefore keeping it online throughout the protocol is unnecessary. Thus, we use it only one-time to generate the system parameters at the starting. From that point onward every user can generate its own key pairs. This allows us to be extremely flexible and scalable in terms of user joining. Note that, similar to [20], no predetermined universal keyword set is demanded and any keyword could be encrypted or searched, thereby increasing the system scalability and flexibility further.

The above description naturally leads to the threat models we consider. Firstly we want a ciphertext hides sender information, receiver-set information and the keyword that it encrypts. We also want a trapdoor should hide sender information, receiver information and the keyword that it encodes. To put it more formally, we want the semi-honest cloud server (or any other unauthorized users) which tests a ciphertext and a trapdoor for *match* should not learn any information about the user-information and keyword encoded in the ciphertext and in the trapdoor. In the next section, we introduce formal security models to capture these threat models. To attain non-triviality, adversaries in our security are restricted by natural restrictions. As per our knowledge, our security models given next considered modeling such strong adversaries for the first time.

3.3 Security Model

We highlight some shortcomings of the security models in [20]. We follow this with new security models for BAEKS.

1. The notion of consistency [1] is quite standard requirement for a searchable encryption. However, [20] did not show the level of consistency it achieved.
2. All the four security games of [20] are (somewhat) non-adaptive. More precisely, the users (sender and receiver(s)) involved in the challenge are set before any trapdoor queries.
3. All the four security games of [20] are (somewhat) one-way. More precisely, the users (sender and receiver(s)) involved in the challenge are chosen by the experiment.
4. All four security games of [20] restrict queries (for both ciphertexts and trapdoors) on the chosen users only.
5. The *trapdoor privacy* was further weakened to allow deterministic trapdoors.
6. Both *ciphertext anonymity* and *trapdoor anonymity* in [20] are very weak as they consider receiver privacy alone, but the privacy of the sender was never considered.
7. Moreover, [20] did not provide any argument for neither *ciphertext anonymity* nor for *trapdoor anonymity*.

For completeness, we provide the descriptions of the security models of [20] verbatim in Appendix A. A savvy reader might want to corroborate the above shortcomings with the games recalled over there.

Adaptive Security Model. Our new security models address the shortcomings mentioned above. Firstly, we define two security models where one captures ciphertext security while the other captures trapdoors' security. We emphasize that both security models capture adaptive semi-honest adversaries. The *trapdoor security* aims to hide the sender information, the receiver information, and the keyword encoded in the trapdoor, and the *ciphertext security* aims to hide the same information encrypted in the ciphertext.

Trapdoor Security. A BAEKS scheme BaEKS satisfies trapdoor security ($\mathsf{trap\text{-}cpa}$) if for all ppt adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{trap\text{-}cpa}}_{\mathcal{A},\mathsf{BaEKS}}(\lambda) = \mathsf{neg}(\lambda)$ where

$$\mathsf{Adv}^{\mathsf{trap\text{-}cpa}}_{\mathcal{A},\mathsf{BaEKS}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{trap\text{-}cpa}}_{\mathsf{BaEKS}}(1^\lambda, 0, \mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{trap\text{-}cpa}}_{\mathsf{BaEKS}}(1^\lambda, 1, \mathcal{A}) = 1]|$$

where $\mathsf{Exp}^{\mathsf{trap\text{-}cpa}}_{\mathsf{BaEKS}}(1^\lambda, \mathfrak{b}, \mathcal{A})$ is defined in Fig. 1.

O_{Ct} is an oracle that on input $(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$ outputs $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{sk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$ after storing $\{(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathsf{R}})\}_{\mathsf{R}\in\mathcal{R}}$ in Q_{Ct}; O_{Tr} is an oracle that on input $(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathsf{R}})$ outputs $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{sk}_{\mathsf{R}})$ after storing $(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathsf{R}})$ in Q_{Tr}; O_{pk} is an oracle that on input j outputs $\mathsf{KeyGen}(\mathsf{pp}, j)$ after storing (j, pk_j) in Q_{pk}. We implicitly assume pk_j has been queried before making a trapdoor or a ciphertext query or the challenge involving user j.

```
$\mathsf{Exp}^{\mathsf{trap\text{-}cpa}}_{\mathsf{BaEKS}}(1^\lambda, \mathfrak{b}, \mathcal{A})$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$Q_{\mathsf{Ct}} \leftarrow \phi,\ Q_{\mathsf{Tr}} \leftarrow \phi,\ Q_{\mathsf{pk}} \leftarrow \phi$
$(\mathsf{pk}_{\tilde{\mathsf{S}}^*}, \tilde{\omega}^*, \mathsf{pk}_{\mathsf{R}^*}) \leftarrow \mathcal{A}^{O_{\mathsf{Ct}}(\cdot,\cdot,\cdot), O_{\mathsf{Tr}}(\cdot,\cdot,\cdot), O_{\mathsf{pk}}(\cdot,\cdot)}$
$\mathsf{Tr}^*_0 \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\tilde{\mathsf{S}}^*}, \tilde{\omega}^*, \mathsf{sk}_{\mathsf{R}^*})$
$\mathsf{Tr}^*_1 \hookleftarrow \mathcal{T}$
$\mathfrak{b}' \leftarrow \mathcal{A}^{O_{\mathsf{Ct}}(\cdot,\cdot,\cdot), O_{\mathsf{Tr}}(\cdot,\cdot,\cdot), O_{\mathsf{pk}}(\cdot,\cdot)}(\mathsf{Tr}^*_{\mathfrak{b}})$
Return $\mathfrak{b}' \wedge (\mathsf{pk}_{\tilde{\mathsf{S}}^*}, \tilde{\omega}^*, \mathsf{pk}_{\mathsf{R}^*}) \notin Q_{\mathsf{Ct}}$
```

Fig. 1. Experiment $\mathsf{trap\text{-}cpa}$

```
$\mathsf{Exp}^{\mathsf{ct\text{-}cpa}}_{\mathsf{BaEKS}}(1^\lambda, \mathfrak{b}, \mathcal{A})$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$Q_{\mathsf{Ct}} \leftarrow \phi,\ Q_{\mathsf{Tr}} \leftarrow \phi,\ Q_{\mathsf{pk}} \leftarrow \phi$
$(\mathsf{pk}_{\mathsf{S}^*}, \omega^*, \mathsf{pk}_{\mathcal{R}^*}) \leftarrow \mathcal{A}^{O_{\mathsf{Ct}}(\cdot,\cdot,\cdot), O_{\mathsf{Tr}}(\cdot,\cdot,\cdot), O_{\mathsf{pk}}(\cdot,\cdot)}$
Let $\mathsf{pk}_{\mathcal{R}^*} = \{\mathsf{pk}_{\mathsf{R}^*_1}, \ldots, \mathsf{pk}_{\mathsf{R}^*_\ell}\}$
$\mathsf{Ct}^*_0 \leftarrow \mathsf{SrchEnc}(\mathsf{pp}, \mathsf{sk}_{\mathsf{S}^*}, \omega^*, \mathsf{pk}_{\mathcal{R}^*})$
$\mathsf{Ct}^*_1 \hookleftarrow \mathcal{CT}$
$\mathfrak{b}' \leftarrow \mathcal{A}^{O_{\mathsf{Ct}}(\cdot,\cdot,\cdot), O_{\mathsf{Tr}}(\cdot,\cdot,\cdot), O_{\mathsf{pk}}(\cdot,\cdot)}(\mathsf{Ct}^*_{\mathfrak{b}})$
Return $\mathfrak{b}' \wedge (\mathsf{pk}_{\mathsf{S}^*}, \omega^*, \mathsf{pk}_{\mathsf{R}^*_i}) \notin Q_{\mathsf{Tr}}$
                                        $\wedge \mathsf{pk}_{\mathsf{R}^*_i} \in \mathsf{pk}_{\mathcal{R}^*}$
```

Fig. 2. Experiment $\mathsf{ct\text{-}cpa}$

Ciphertext Security. A BAEKS scheme BaEKS satisfies ciphertext security ($\mathsf{ct\text{-}cpa}$) if for all ppt adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ct\text{-}cpa}}_{\mathcal{A},\mathsf{BaEKS}}(\lambda) = \mathsf{neg}(\lambda)$ for

$$\mathsf{Adv}^{\mathsf{ct\text{-}cpa}}_{\mathcal{A},\mathsf{BaEKS}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{ct\text{-}cpa}}_{\mathsf{BaEKS}}(1^\lambda, 0, \mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ct\text{-}cpa}}_{\mathsf{BaEKS}}(1^\lambda, 1, \mathcal{A}) = 1]|$$

where $\mathsf{Exp}^{\mathsf{ct\text{-}cpa}}_{\mathsf{BaEKS}}(1^\lambda, \mathfrak{b}, \mathcal{A})$ is defined in Fig. 2. The oracle $O_{\mathsf{Ct}}, O_{\mathsf{Tr}}, O_{\mathsf{pk}}$ in Fig. 2 are defined the same as they were in the case of Fig. 1.

4 A Construction of Broadcast Authenticated Encryption with Keyword Search

As we explained in Introduction, BaEKS below defines a ciphertext using $|\mathcal{R}|$ many Boneh-Boyen Hash $(\mathbf{U}_{\mathsf{S}}^\top \cdot \tilde{\omega} + \mathbf{V}_{\mathsf{R}}^\top)\mathbf{A}$ where $\mathbf{U}_{\mathsf{S}}\mathbf{A}$ is sender's secret key and $\mathbf{V}_{\mathsf{R}}\mathbf{A}$ is receiver's public key and defines a trapdoor as a Boneh-Boyen [4] secret key $\boldsymbol{\alpha}_{\mathsf{R}} + (\mathbf{U}_{\tilde{\mathsf{S}}} \cdot \tilde{\omega} + \mathbf{V}_{\mathsf{R}})\mathbf{B}$ as a hash proof system where $\mathbf{U}_{\tilde{\mathsf{S}}}\mathbf{B}$ is sender's public key and $\boldsymbol{\alpha}_{\mathsf{R}}, \mathbf{V}_{\mathsf{R}}\mathbf{B}$ are receiver's secret key. Test essentially checks if any of the $|\mathcal{R}|$ ciphertext components *matches* with the given trapdoor.

- $\mathsf{Setup}(1^\lambda)$:
  1. Sample $\mathbf{A}, \mathbf{B} \hookleftarrow \mathcal{D}_k$.
  2. Publish $\mathsf{pp} = (\llbracket \mathbf{A} \rrbracket_1, \llbracket \mathbf{A} \rrbracket_2, \llbracket \mathbf{B} \rrbracket_1, \llbracket \mathbf{B} \rrbracket_2)$.
- $\mathsf{KeyGen}(\mathsf{pp}, j)$:
  1. Sample $\boldsymbol{\alpha}_j \hookleftarrow \mathbb{Z}_p^{(k+1)}$.
  2. Sample $\mathbf{U}_j, \mathbf{V}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$.
  3. Set $\mathsf{sk}_j = (\llbracket \boldsymbol{\alpha}_j \rrbracket_2, \left\llbracket \mathbf{U}_j^\top \mathbf{A} \right\rrbracket_1, \llbracket \mathbf{V}_j \mathbf{B} \rrbracket_2)$.
  4. Publish $\mathsf{pk}_j = (\left\llbracket \boldsymbol{\alpha}_j^\top \mathbf{A} \right\rrbracket_{\mathrm{T}}, \llbracket \mathbf{U}_j \mathbf{B} \rrbracket_2, \left\llbracket \mathbf{V}_j^\top \mathbf{A} \right\rrbracket_1)$.
- $\mathsf{SrchEnc}(\mathsf{pp}, \mathsf{sk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$:
  1. Let $\mathsf{pk}_{\mathcal{R}} = (\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell})$ where $\mathcal{R} = (\mathsf{R}_1, \ldots, \mathsf{R}_\ell)$.
  2. Sample $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$ and a random permutation $\tau : [\ell] \to [\ell]$.
  3. Compute $\mathsf{ct}_0 = \llbracket \mathbf{As} \rrbracket_1$, $\mathsf{ct}_{1,j} = \left\llbracket (\omega \cdot \mathbf{U}_{\mathsf{S}}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{As} \right\rrbracket_1$ and $\kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top \mathbf{As} \right\rrbracket_{\mathrm{T}}$ for all $j \in [\ell]$.
  4. Output $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,1}, \kappa_1), \ldots, (\mathsf{ct}_{1,\ell}, \kappa_\ell))$.
- $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\widetilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{sk}_{\mathsf{R}})$:
  1. Sample $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$.
  2. Compute $\mathsf{tr}_0 = \llbracket \mathbf{Br} \rrbracket_2$, $\mathsf{tr}_1 = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}} + (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{Br} \right\rrbracket_2$.
  3. Output $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$.
- $\mathsf{Test}(\mathsf{Tr}, \mathsf{Ct})$:
  1. Let $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,1}, \kappa_1), \ldots, (\mathsf{ct}_{1,\ell}, \kappa_\ell))$.
  2. Parse $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$.

$$\text{Output} = \begin{cases} 1 & \text{If } \; e(\mathsf{ct}_0, \mathsf{tr}_1) = \kappa_j \cdot e(\mathsf{ct}_{1,j}, \mathsf{tr}_0),\ \exists j \in [\ell] \\ 0 & \text{Otherwise} \end{cases}.$$

Correctness. Suppose, $\widetilde{\mathsf{S}} = \mathsf{S}$, $\tilde{\omega} = \omega$ and $\mathsf{R} \in \mathcal{R} = \{\mathsf{R}_1, \ldots, \mathsf{R}_\ell\}$ for

- $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,1}, \kappa_1), \ldots, (\mathsf{ct}_{1,\ell}, \kappa_\ell)) \leftarrow \mathsf{SrchEnc}(\mathsf{pp}, \mathsf{sk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$
- $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1) \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\widetilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{sk}_{\mathsf{R}})$,

$$\begin{aligned} A &= e(\mathsf{ct}_0, \mathsf{tr}_1) = e(\llbracket \mathbf{As} \rrbracket_1, \llbracket \boldsymbol{\alpha}_{\mathsf{R}} + (\omega \cdot \mathbf{U}_{\mathsf{S}} + \mathbf{V}_{\mathsf{R}})\mathbf{Br} \rrbracket_2) \\ &= \llbracket \boldsymbol{\alpha}_{\mathsf{R}}^\top \mathbf{As} \rrbracket_{\mathrm{T}} \cdot e(\left\llbracket (\omega \cdot \mathbf{U}_{\mathsf{S}}^\top + \mathbf{V}_{\mathsf{R}}^\top)\mathbf{As} \right\rrbracket_1, \llbracket \mathbf{Br} \rrbracket_2) \\ B &= \kappa_j \cdot e(\mathsf{ct}_{1,j}, \mathsf{tr}_0) = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top \mathbf{As} \right\rrbracket_{\mathrm{T}} \cdot e(\left\llbracket (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{As} \right\rrbracket_1, \llbracket \mathbf{Br} \rrbracket_2) \\ \frac{A}{B} &= \left\llbracket (\boldsymbol{\alpha}_{\mathsf{R}}^\top - \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{As} \right\rrbracket_{\mathrm{T}} \cdot \left\llbracket \mathbf{r}^\top \mathbf{B}^\top (\omega \cdot \mathbf{U}_{\mathsf{S}}^\top - \tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}}^\top + \mathbf{V}_{\mathsf{R}}^\top - \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{As} \right\rrbracket_{\mathrm{T}} \end{aligned}$$

Since, $\widetilde{\mathsf{S}} = \mathsf{S}$, $\tilde{\omega} = \omega$ and $\mathsf{R} \in \mathcal{R}$ ($\exists j \in [\ell]$, $\mathsf{R} = \mathsf{R}_{\tau(j)}$), $\Pr[A = B] = 1$.

Consistency. Informally speaking, we need to show that for distinct $i, j \in \{0, 1\}$, if $\mathsf{S}_j \neq \mathsf{S}_i$ or if $\omega_j \neq \omega_i$ or if $\mathsf{R}_j \notin \mathcal{R}_i$ for adversarially chosen keywords $\omega_j, \omega_i \in \mathcal{KW}$, a trapdoor $\mathsf{Tr}_j \leftarrow \mathsf{Trapdoor}(\mathsf{pk}_{\mathsf{S}_j}, \omega_j, \mathsf{sk}_{\mathsf{R}_j})$ *matches* a ciphertext $\mathsf{Ct}_i \leftarrow \mathsf{SrchEnc}(\mathsf{sk}_{\mathsf{S}_i}, \omega_i, \mathsf{pk}_{\mathcal{R}_i})$ with only negligible probability. We emphasize that this holds even against statistical adversary that is given public keys of the senders and the receivers.

Proof Sketch. Observe that adversary $\mathcal{A}$ gets params $\mathsf{pp} = (\llbracket \mathbf{A} \rrbracket_1, \llbracket \mathbf{A} \rrbracket_2, \llbracket \mathbf{B} \rrbracket_1, \llbracket \mathbf{B} \rrbracket_2)$. $\mathcal{A}$ also gets the public keys for $\big((\mathsf{pk}_t)_{t \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{R}_0, \mathsf{R}_1\}}\big)$. It therefore has access to $\mathsf{pk}_t = (\llbracket \boldsymbol{\alpha}_t^\top \mathbf{A} \rrbracket_\mathrm{T}, \llbracket \mathbf{U}_t \mathbf{B} \rrbracket_2, \llbracket \mathbf{V}_t^\top \mathbf{A} \rrbracket_1)$ where $\boldsymbol{\alpha}_t \hookleftarrow \mathbb{Z}_p^{(k+1)}, \mathbf{U}_t, \mathbf{V}_t \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$ are sampled uniformly at random by the experiment for all $t \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{R}_0, \mathsf{R}_1\}$. We show that, given all these information, there is enough entropy in the corresponding secret keys $\big((\mathsf{sk}_t)_{t \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{R}_0, \mathsf{R}_1\}}\big)$ to show that the best $\mathcal{A}$ can do is to guess despite $\mathcal{A}$ being an unbounded adversary. □

To formulate our argument, we sample $\mathbf{U}_t = \widetilde{\mathbf{U}}_t + \widetilde{u}_t \mathbf{T}$ and $\mathbf{V}_t = \widetilde{\mathbf{V}}_t + \widetilde{v}_t \mathbf{T}$ for $\widetilde{\mathbf{U}}_t, \widetilde{\mathbf{V}}_t, \mathbf{T} \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$, $\widetilde{u}_t, \widetilde{v}_t \hookleftarrow \mathbb{Z}_p$ for all $t \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{R}_0, \mathsf{R}_1\}$. Since, $\widetilde{\mathbf{U}}_t$ and $\widetilde{\mathbf{V}}_t$ are uniformly random, $\mathbf{U}_t$ and $\mathbf{V}_t$ are uniformly random for all $t \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{R}_0, \mathsf{R}_1\}$. Thus, for all $t \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{R}_0, \mathsf{R}_1\}$, pk_t are properly distributed and do not leak any information about the corresponding $\widetilde{u}_t$ and $\widetilde{v}_t$. We sample $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$. The trapdoor Tr_j is now $\mathsf{Tr}_j = (\mathsf{tr}_0^{[j]} = \llbracket \mathbf{Br} \rrbracket_2, \mathsf{tr}_1^{[j]} = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_j} + (\omega_j \cdot \widetilde{\mathbf{U}}_{\mathsf{S}_j} + \widetilde{\mathbf{V}}_{\mathsf{R}_j})\mathbf{Br} + \boxed{(\omega_j \widetilde{u}_{\mathsf{S}_j} + \widetilde{v}_{\mathsf{R}_j})} \mathbf{TBr} \right\rrbracket_2)$. On the other hand, $\mathsf{Ct}_i = (\mathsf{ct}_0^{[i]}, (\mathsf{ct}_{1,l}^{[i]}, \kappa_l^{[i]})_{l \in [\ell]})$ for $\mathcal{R}_i = \{\mathsf{R}_1^{[i]}, \ldots, \mathsf{R}_\ell^{[i]}\}$, a random permutation $\tau : [\ell] \to [\ell]$ and $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$ such that,

$$\begin{aligned}
\mathsf{ct}_0^{[i]} &= \llbracket \mathbf{As} \rrbracket_1 \\
\mathsf{ct}_{1,l}^{[i]} &= \left\llbracket (\omega_i \cdot \mathbf{U}_{\mathsf{S}_i} + \mathbf{V}^\top_{\mathsf{R}^{[i]}_{\tau(l)}})\mathbf{As} + \boxed{(\omega_i \widetilde{u}_{\mathsf{S}_i} + \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(l)}})} \mathbf{T}^\top \mathbf{As} \right\rrbracket_1 . \\
\kappa_l^{[i]} &= \left\llbracket \boldsymbol{\alpha}^\top_{\mathsf{R}^{[i]}_{\tau(l)}} \mathbf{As} \right\rrbracket_\mathrm{T}
\end{aligned}$$

We now focus at the boxed part of the above equations. Let $\sigma_l = (\omega_i \widetilde{u}_{\mathsf{S}_i} + \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(l)}})$ for all $l \in [\ell]$ and $\sigma_{\ell+1} = (\omega_j \widetilde{u}_{\mathsf{S}_j} + \widetilde{v}_{\mathsf{R}_j})$.

Case-1 Consider the case $\mathsf{S}_j \neq \mathsf{S}_i$.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{\ell+1} \end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix} \omega_i & 0 & 1 & 0 & \cdots & 0 & 0 \\ \omega_i & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \omega_i & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & \omega_j & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}_{(\ell+1)\times(\ell+3)} \times \begin{bmatrix} \widetilde{u}_{\mathsf{S}_i} \\ \widetilde{u}_{\mathsf{S}_j} \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(1)}} \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(2)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(\ell)}} \\ \widetilde{v}_{\mathsf{R}_j} \end{bmatrix}_{(\ell+3)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell + 1)$. Since, $\widetilde{u}_{\mathsf{S}_i}, \widetilde{u}_{\mathsf{S}_j}, \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(\ell)}}, \widetilde{v}_{\mathsf{R}_j}$ are sampled uniformly at random, $\sigma_1, \ldots, \sigma_{\ell+1}$ are too independent and uniformly random.

Case-2 Consider the case $\mathsf{S}_j = \mathsf{S}_i \wedge \omega_j \neq \omega_i$.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{\ell+1} \end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix} \omega_i & 1 & 0 & \cdots & 0 & 0 \\ \omega_i & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \omega_i & 0 & 0 & \cdots & 1 & 0 \\ \omega_j & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}_{(\ell+1)\times(\ell+2)} \times \begin{bmatrix} \widetilde{u}_{\mathsf{S}_i} \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(1)}} \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(2)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(\ell)}} \\ \widetilde{v}_{\mathsf{R}_j} \end{bmatrix}_{(\ell+2)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell + 1)$. Since, $\widetilde{u}_{\mathsf{S}_i}, \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(\ell)}}, \widetilde{v}_{\mathsf{R}_j}$ are sampled uniformly at random, $\sigma_1, \ldots, \sigma_{\ell+1}$ are too independent and uniformly random.

Case-3 Consider the case $\mathsf{S}_j = \mathsf{S}_i \wedge \omega_j = \omega_i \wedge \mathsf{R}_j \notin \mathcal{R}_i$.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{\ell+1} \end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix} \omega_i & 1 & 0 & \cdots & 0 & 0 \\ \omega_i & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \omega_i & 0 & 0 & \cdots & 1 & 0 \\ \omega_i & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}_{(\ell+1)\times(\ell+2)} \times \begin{bmatrix} \widetilde{u}_{\mathsf{S}_i} \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(1)}} \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(2)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(\ell)}} \\ \widetilde{v}_{\mathsf{R}_j} \end{bmatrix}_{(\ell+2)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell + 1)$. Since, $\widetilde{u}_{\mathsf{S}_i}, \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}^{[i]}_{\tau(\ell)}}, \widetilde{v}_{\mathsf{R}_j}$ are sampled uniformly at random, $\sigma_1, \ldots, \sigma_{\ell+1}$ are too independent and uniformly random.

In other words, $\sigma_1, \ldots, \sigma_{\ell+1}$ are completely uncorelated and as a result Ct_i and Tr_j considered in the consistency game *does not match* except with negligible probability.

Security. Next, we argue that our construction BaEKS is adaptively secure.

Theorem 1. *If the bil-$\mathcal{D}_k$-*matDH *Assumption holds in $\mathcal{G}$, then* BaEKS *is a* trap-cpa*-secure BAEKS scheme. Precisely, for any* ppt *adversary $\mathcal{A}$ that breaks* BaEKS *in the* trap-cpa *model making at most q trapdoor queries, at most Q encryption requests and at most $\tilde{Q}$ public key requests, there exists a* ppt *algorithm $\mathcal{B}$ such that,*

$$\mathsf{Adv}^{\mathsf{trap\text{-}cpa}}_{\mathcal{A},\mathsf{BaEKS}}(\lambda) \leq (2Q + 1) \cdot \mathsf{Adv}^{bil\text{-}\mathcal{D}_k\text{-}\mathsf{matDH}}_{\mathcal{B},\mathcal{G}}(\lambda).$$

Theorem 2. *If the bil-$\mathcal{D}_k$-*matDH *Assumption holds in* $\mathcal{G}$*, then* BaEKS *is a* $\mathsf{ct\text{-}cpa}$*-secure BAEKS scheme. Precisely, for any* ppt *adversary* $\mathcal{A}$ *that breaks* BaEKS *in the* $\mathsf{ct\text{-}cpa}$ *model making at most* q *trapdoor queries, at most* Q *encryption requests and at most* $\tilde{Q}$ *public key requests, there exists a* ppt *algorithm* $\mathcal{B}$ *such that,*

$$\mathsf{Adv}^{\mathsf{ct\text{-}cpa}}_{\mathcal{A},\mathsf{BaEKS}}(\lambda) \leq (2q+1)\cdot \mathsf{Adv}^{bil\text{-}\mathcal{D}_k\text{-}\mathsf{matDH}}_{\mathcal{B},\mathcal{G}}(\lambda).$$

4.1 Trapdoor Security

Proof of Theorem 1. We give the proof via a hybrid argument similar to the dual system proof technique [22]. In this proof technique, the ciphertexts and challenge trapdoors can be of two kinds: (*i*) normal and (*ii*) semi-functional. Note that a semi-functional trapdoor cannot *match* a semi-functional ciphertext even if they encode *matching* attributes.

We informally describe the sequence of games here. Game_0 is the real security game where the challenge trapdoor, all the queried ciphertexts, and trapdoors are normal. The challenge trapdoor is made semi-functional in Game_1. Following subsequence of games $(\mathsf{Game}_{2,i,1}, \mathsf{Game}_{2,i,2}, \mathsf{Game}_{2,i,3})_{i\in[Q]}$ make the queried ciphertexts semi-functional. In particular, in $\mathsf{Game}_{2,i,j}$ the i^{th} ciphertext is type-j semi-functional. Finally, in Game_3, we replace the challenge trapdoor $\mathsf{Tr}^*_0 = (\mathsf{tr}^{[0]}_0, \mathsf{tr}^{[0]}_1)$ by independent random choices. Note that, we denote Game_1 by $\mathsf{Game}_{2,0}$ as well. The indistinguishability of Game_0 and Game_3 is proven via the sequence of games mentioned in Table 1.

Table 1. Outline of the proof strategy

Games	Difference from	Indistinguishability	
	Previous Game	from Previous Game	under Assumption
Game_0	–	–	
Game_1	challenge trapdoor Tr^*_0 is semi-functional	Lemma 1	*bil*-$\mathcal{D}_k$-matDH
$\mathsf{Game}_{2,i,1}$	i^{th} ciphertext is type-1 semi-functional	Lemma 2	*bil*-$\mathcal{D}_k$-matDH
$\mathsf{Game}_{2,i,2}$	i^{th} ciphertext is type-2 semi-functional	Lemma 3	information theoretic
$\mathsf{Game}_{2,i,3}$	i^{th} ciphertext is type-3 semi-functional	Lemma 4	*bil*-$\mathcal{D}_k$-matDH
Game_3	challenge trapdoor Tr^*_0 is random	Lemma 5	information theoretic

4.1.1 Semi-functional Algorithms

- $\mathsf{sfSrchEnc}(\mathsf{pp}, \mathsf{sk}_\mathsf{S}, \omega, \mathsf{sk}_\mathcal{R})$: Let $\mathsf{sk}_\mathcal{R} = \{\mathsf{sk}_{\mathsf{R}_1}, \ldots, \mathsf{sk}_{\mathsf{R}_\ell}\}$. Choose $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$, $\widehat{u}_1, \ldots, \widehat{u}_\ell, \widehat{s} \hookleftarrow \mathbb{Z}_p$ and $\mathbf{b}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{b}^\perp}^\top \mathbf{B} = 0$. Compute the semi-functional ciphertext $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$ as follows:
  - $\mathsf{ct}_0 = \left[\!\left[\mathbf{A}\mathbf{s} + \delta \mathbf{b}^\perp \widehat{s}\right]\!\right]_1$,
  - $\mathsf{ct}_{1,j} = \left[\!\left[\nu \mathbf{b}^\perp \widehat{u}_j + (\omega \cdot \mathbf{U}_\mathsf{S}^\top + \mathbf{V}^\top_{\mathsf{R}_{\tau(j)}})(\mathbf{A}\mathbf{s} + \delta \mathbf{b}^\perp \widehat{s})\right]\!\right]_1$,

- $\kappa_j = \left[\!\left[\boldsymbol{\alpha}^\top_{\mathsf{R}_{\tau(j)}} (\mathbf{A}\mathbf{s} + \delta \mathbf{b}^\perp \widehat{s}) \right]\!\right]_{\mathrm{T}}$,

  where if $(\nu = 0, \delta = 1)$, Ct is a type-1 semi-functional ciphertext; if $(\nu = 1, \delta = 1)$, Ct is a type-2 semi-functional ciphertext; and if $(\nu = 1, \delta = 0)$, Ct is a type-3 semi-functional ciphertext.

– $\mathsf{sfTrapdoor}(\mathsf{pp}, \mathsf{sk}_{\tilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{sk}_{\mathsf{R}})$: Choose $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$, $\widehat{r} \hookleftarrow \mathbb{Z}_p$ and $\mathbf{a}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^\perp}^\top \mathbf{A} = 0$. Compute the semi-functional trapdoor $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$ as follows.

$$(\mathsf{tr}_0, \mathsf{tr}_1) = \left([\![\mathbf{B}\mathbf{r} + \mathbf{a}^\perp \widehat{r}]\!]_2 , [\![\boldsymbol{\alpha}_{\mathsf{R}} + (\tilde{\omega} \cdot \mathbf{U}_{\tilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})(\mathbf{B}\mathbf{r} + \mathbf{a}^\perp \widehat{r})]\!]_2 \right).$$

4.1.2 Sequence of Games

Lemma 1. (Game_0 *to* Game_1) *For any* ppt *adversary* $\mathcal{A}$ *that makes at most* q *trapdoor queries, at most* Q *encryption requests, and at most* $\tilde{Q}$ *public key requests; there exists a* ppt *adversary* $\mathcal{B}$ *such that,*

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_0}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_1}(\lambda)| \leq \mathsf{Adv}_{\mathcal{B}}^{bil\text{-}\mathcal{D}_k\text{-}\mathsf{matDH}}.$$

Lemma 2. ($\mathsf{Game}_{2,i-1,3}$ *to* $\mathsf{Game}_{2,i,1}$) *For any* ppt *adversary* $\mathcal{A}$ *making at most* q *trapdoor queries, at most* Q *encryption requests, and at most* $\tilde{Q}$ *public key requests; there exists a* ppt *adversary* $\mathcal{B}$ *such that,*

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i-1,3}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,1}}(\lambda)| \leq \mathsf{Adv}_{\mathcal{B}}^{bil\text{-}\mathcal{D}_k\text{-}\mathsf{matDH}}.$$

Lemma 3. ($\mathsf{Game}_{2,i,1}$ *to* $\mathsf{Game}_{2,i,2}$) *For any adversary* $\mathcal{A}$*,*

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,1}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,2}}(\lambda)| = 0.$$

Lemma 4. ($\mathsf{Game}_{2,i,2}$ *to* $\mathsf{Game}_{2,i,3}$) *For any* ppt *adversary* $\mathcal{A}$ *that makes at most* q *trapdoor queries, at most* Q *encryption requests, and at most* $\tilde{Q}$ *public key requests; there exists a* ppt *adversary* $\mathcal{B}$ *such that,*

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,2}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,3}}(\lambda)| \leq \mathsf{Adv}_{\mathcal{B}}^{bil\text{-}\mathcal{D}_k\text{-}\mathsf{matDH}}.$$

Lemma 5. ($\mathsf{Game}_{2,Q,3}$ *to* Game_3) *For any adversary* $\mathcal{A}$*,*

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,Q,3}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_3}(\lambda)| = 0.$$

Lemma 6. (Game_3) *For any adversary* $\mathcal{A}$*,* $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_3}(\lambda)| = 0$.

4.1.3 Proof of Games

Proof of Lemma 1. $\mathcal{B}$ receives a $bil\text{-}\mathcal{D}_k$-matDH problem instance $([\![\mathbf{N}]\!]_1, [\![\mathbf{y}]\!]_1, [\![\mathbf{N}]\!]_2, [\![\mathbf{y}]\!]_2)$, where $\mathbf{y} = \mathbf{N}\mathbf{z}$ or $\mathbf{y} \hookleftarrow \mathbb{Z}_p^{k+1}$.

– **Setup**: $\mathcal{B}$ samples $\mathfrak{b} \hookleftarrow \{0,1\}$, $\mathbf{A} \leftarrow \mathcal{D}_k$ and $\mathbf{a}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^\perp}^\top \mathbf{A} = \mathbf{0}$. It publishes $\mathsf{pp} = ([\![\mathbf{A}]\!]_1, [\![\mathbf{A}]\!]_2, [\![\mathbf{N}]\!]_1, [\![\mathbf{N}]\!]_2)$.

- **Query Phase-**I: $\mathcal{A}$ makes following queries–
  - $O_{\mathsf{pk}}(j)$: $\mathcal{B}$ samples $\boldsymbol{\alpha}_j \hookleftarrow \mathbb{Z}_p^{k+1}$, $\mathbf{U}_j, \mathbf{V}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$. It returns $\mathsf{pk}_j = (\llbracket \boldsymbol{\alpha}_j^\top \rrbracket_{\mathrm{T}}, \llbracket \mathbf{U}_j\mathbf{N} \rrbracket_2, \left\llbracket \mathbf{V}_j^\top \mathbf{A} \right\rrbracket_1)$.
  - $O_{\mathsf{Tr}}(\mathsf{pk}_{\tilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{pk}_{\mathsf{R}})$: $\mathcal{B}$ samples $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$. It outputs $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$, where $\mathsf{tr}_0 = \llbracket \mathbf{Nr} \rrbracket_2, \mathsf{tr}_1 = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}} + (\tilde{\omega} \cdot \mathbf{U}_{\tilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{Nr} \right\rrbracket_2$.
  - $O_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$: $\mathcal{B}$ samples $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$, defines a random permutation $\tau : [\ell] \to [\ell]$ for $\mathsf{pk}_{\mathcal{R}} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$. It outputs $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$ where

$$\mathsf{ct}_0 = \llbracket \mathbf{As} \rrbracket_1, \mathsf{ct}_{1,j} = \left\llbracket (\omega \cdot \mathbf{U}_{\mathsf{S}}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{As} \right\rrbracket_1, \kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top \mathbf{As} \right\rrbracket_{\mathrm{T}}.$$

- **Challenge**: $\mathcal{A}$ makes a challenge on $(\mathsf{pk}_{\tilde{\mathsf{S}}^*}, \tilde{\omega}^*, \mathsf{pk}_{\mathsf{R}^*})$. $\mathcal{B}$ computes $\mathsf{Tr}_0^* = (\mathsf{tr}_0^{[0]}, \mathsf{tr}_1^{[0]})$ where $\mathsf{tr}_0^{[0]} = \llbracket \mathbf{y} \rrbracket_2$ and $\mathsf{tr}_1^{[0]} = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}^*} + (\tilde{\omega}^* \mathbf{U}_{\tilde{\mathsf{S}}^*} + \mathbf{V}_{\mathsf{R}^*})\mathbf{y} \right\rrbracket_2$ and samples $\mathsf{Tr}_1^* \hookleftarrow \mathcal{T}$. It returns $\mathsf{Tr}_{\mathfrak{b}}^*$ to $\mathcal{A}$.
- **Query Phase-**II: Same as **Query Phase-**I.
- **Guess**: $\mathcal{B}$ forwards $\mathfrak{b}'$ that is output by $\mathcal{A}$.

It is easy to see that if $\mathbf{y} = \mathbf{Nz}$, Tr_0^* is normal; and if $\mathbf{y} \hookleftarrow \mathbb{Z}_p^{k+1}$, Tr_0^* is semi-functional. □

Proof of Lemma 2. $\mathcal{B}$ receives a *bil-*$\mathcal{D}_k$-matDH problem instance $(\llbracket \mathbf{M} \rrbracket_1, \llbracket \mathbf{z} \rrbracket_1, \llbracket \mathbf{M} \rrbracket_2, \llbracket \mathbf{z} \rrbracket_2)$, where $\mathbf{z} = \mathbf{My}$ or $\mathbf{z} \hookleftarrow \mathbb{Z}_p^{k+1}$.

- **Setup**: $\mathcal{B}$ samples $\mathfrak{b} \hookleftarrow \{0,1\}, \mathbf{B} \leftarrow \mathcal{D}_k$ and $\mathbf{b}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that $\mathbf{b}^{\perp\top}\mathbf{B} = \mathbf{0}$. It publishes $\mathsf{pp} = (\llbracket \mathbf{M} \rrbracket_1, \llbracket \mathbf{M} \rrbracket_2, \llbracket \mathbf{B} \rrbracket_1, \llbracket \mathbf{B} \rrbracket_2)$.
- **Query Phase-**I: $\mathcal{A}$ makes following queries–
  - $O_{\mathsf{pk}}(j)$: $\mathcal{B}$ samples $\boldsymbol{\alpha}_j \hookleftarrow \mathbb{Z}_p^{k+1}$, $\mathbf{U}_j, \mathbf{V}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$. It returns $\mathsf{pk}_j = (\llbracket \boldsymbol{\alpha}_j^\top \rrbracket_{\mathrm{T}}, \llbracket \mathbf{U}_j\mathbf{B} \rrbracket_2, \left\llbracket \mathbf{V}_j^\top \mathbf{M} \right\rrbracket_1)$.
  - $O_{\mathsf{Tr}}(\mathsf{pk}_{\tilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{pk}_{\mathsf{R}})$: $\mathcal{B}$ samples $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$. It outputs $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$, where $\mathsf{tr}_0 = \llbracket \mathbf{Br} \rrbracket_2, \mathsf{tr}_1 = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}} + (\tilde{\omega} \cdot \mathbf{U}_{\tilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{Br} \right\rrbracket_2$.
  - $O_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$: On t^{th} query, $\mathcal{B}$ first samples a random permutation $\tau : [\ell] \to [\ell]$ for $\mathsf{pk}_{\mathcal{R}} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$ and does the following:
    - $t > i$**:** It samples $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$ to output $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$:

$$\mathsf{ct}_0 = \llbracket \mathbf{Ms} \rrbracket_1, \mathsf{ct}_{1,j} = \left\llbracket (\omega \cdot \mathbf{U}_{\mathsf{S}}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{Ms} \right\rrbracket_1, \kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top \mathbf{Ms} \right\rrbracket_{\mathrm{T}}.$$

    - $t < i$**:** To output $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$, it samples $\mathbf{s}, \widehat{u}_{i,1}, \ldots, \widehat{u}_{i,\ell} \hookleftarrow \mathbb{Z}_p^k$:

$$\mathsf{ct}_0 = \llbracket \mathbf{Ms} \rrbracket_1, \mathsf{ct}_{1,j} = \left\llbracket \mathbf{b}^\perp \widehat{u}_{i,j} + (\omega \cdot \mathbf{U}_{\mathsf{S}}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{Ms} \right\rrbracket_1, \kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top \mathbf{Ms} \right\rrbracket_{\mathrm{T}}.$$

$-t = i$: It outputs $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j \in [\ell]})$ where

$$\mathsf{ct}_0 = \llbracket \mathbf{z} \rrbracket_1, \mathsf{ct}_{1,j} = \left\llbracket (\omega \cdot \mathbf{U}_{\mathsf{S}}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{z} \right\rrbracket_1, \kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top \mathbf{z} \right\rrbracket_{\mathrm{T}}.$$

- **Challenge**: $\mathcal{A}$ makes a challenge on $(\mathsf{pk}_{\widetilde{\mathsf{S}}^*}, \tilde{\omega}^*, \mathsf{pk}_{\mathsf{R}^*})$. $\mathcal{B}$ computes $\mathsf{Tr}_0^* = (\mathsf{tr}_0^{[0]}, \mathsf{tr}_1^{[0]})$ where $\mathsf{tr}_0^{[0]} = \llbracket \widetilde{\mathbf{r}} \rrbracket_2$ and $\mathsf{tr}_1^{[0]} = \llbracket \boldsymbol{\alpha}_{\mathsf{R}^*} + (\tilde{\omega}^* \mathbf{U}_{\widetilde{\mathsf{S}}^*} + \mathbf{V}_{\mathsf{R}^*})\widetilde{\mathbf{r}} \rrbracket_2$ for $\widetilde{\mathbf{r}} \hookleftarrow \mathbb{Z}_p^{k+1}$ and samples $\mathsf{Tr}_1^* \hookleftarrow \mathcal{T}$. It returns $\mathsf{Tr}_{\mathfrak{b}}^*$ to $\mathcal{A}$.
- **Query Phase-**II: Same as **Query Phase-**I.
- **Guess**: $\mathcal{B}$ forwards $\mathfrak{b}'$ that is output by $\mathcal{A}$.

It is easy to see that if $\mathbf{z} = \mathbf{Ny}$, the i^{th} ciphertext response Ct is normal; and if $\mathbf{z} \hookleftarrow \mathbb{Z}_p^{k+1}$, the i^{th} ciphertext response Ct is semi-functional. □

Proof of Lemma 3. Recall that, the challenge trapdoor Tr_0^* encodes $(\widetilde{\mathsf{S}}^*, \tilde{\omega}^*, \mathsf{R}^*)$ that *does not match* $(\mathsf{S}, \omega, \mathcal{R})$ encoded in the i^{th} ciphertext Ct_i. We here argue that, the joint distributions of $\{\mathsf{pp}, \mathsf{PK}, \mathsf{Ct}_i, \mathsf{Tr}_0^*\}$ if Ct_i is a type-1 semi-functional ciphertext *is identical to* the joint distributions of $\{\mathsf{pp}, \mathsf{PK}, \mathsf{Ct}_i, \mathsf{Tr}_0^*\}$ if Ct_i is a type-2 semi-functional ciphertext where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{PK} = \{\mathsf{pk}_j\}_{j \in \mathcal{R} \cup \{\mathsf{S}, \mathsf{R}^*, \widetilde{\mathsf{S}}^*\}}$ for $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, j)$ and $\mathsf{Tr}_0^* = (\mathsf{tr}_0^{[0]}, \mathsf{tr}_1^{[0]}) \leftarrow \mathsf{sfTrapdoor}(\mathsf{sk}_{\widetilde{\mathsf{S}}^*}, \tilde{\omega}^*, \mathsf{sk}_{\mathsf{R}^*})$ is a semi-functional trapdoor.

We sample $\mathbf{A}, \mathbf{B} \leftarrow \mathcal{D}_k$ and $\mathbf{a}^\perp, \mathbf{b}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^\perp}^\top \mathbf{A} = {\mathbf{b}^\perp}^\top \mathbf{B} = 0$ but $\mu = {\mathbf{a}^\perp}^\top \mathbf{b}^\perp \neq 0$. We sample $\mathbf{U}_j, \mathbf{V}_j$ differently for any user j. In particular, we define $\mathbf{U}_j = \widetilde{\mathbf{U}}_j + \widetilde{u}_j \mu^{-1} \mathbf{a}^\perp {\mathbf{b}^\perp}^\top$ and $\mathbf{V}_j = \widetilde{\mathbf{V}}_j + \widetilde{v}_j \mu^{-1} \mathbf{a}^\perp {\mathbf{b}^\perp}^\top$ for $\widetilde{\mathbf{U}}_j, \widetilde{\mathbf{V}}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$ and $\widetilde{u}_j, \widetilde{v}_j \hookleftarrow \mathbb{Z}_p$. Observe from the following equations that pk_j stays the same for all user j.

$$\boxed{\begin{array}{ll}
\mathbf{U}_j^\top \mathbf{A} = \widetilde{\mathbf{U}}_j^\top \mathbf{A} & \mathbf{U}_j^\top \mathbf{b}^\perp = \widetilde{\mathbf{U}}_j^\top \mathbf{b}^\perp + \widetilde{u}_j \mu^{-1} \mathbf{b}^\perp ({\mathbf{a}^\perp}^\top \mathbf{b}^\perp) = \widetilde{\mathbf{U}}_j^\top \mathbf{b}^\perp + \widetilde{u}_j \mathbf{b}^\perp \\
\mathbf{U}_j \mathbf{B} = \widetilde{\mathbf{U}}_j \mathbf{B} & \mathbf{U}_j \mathbf{a}^\perp = \widetilde{\mathbf{U}}_j \mathbf{a}^\perp + \widetilde{u}_j \mu^{-1} \mathbf{a}^\perp ({\mathbf{b}^\perp}^\top \mathbf{a}^\perp) = \widetilde{\mathbf{U}}_j \mathbf{b}^\perp + \widetilde{u}_j \mathbf{a}^\perp \\
\mathbf{V}_j^\top \mathbf{A} = \widetilde{\mathbf{V}}_j^\top \mathbf{A} & \mathbf{V}_j^\top \mathbf{b}^\perp = \widetilde{\mathbf{V}}_j^\top \mathbf{b}^\perp + \widetilde{v}_j \mu^{-1} \mathbf{b}^\perp ({\mathbf{a}^\perp}^\top \mathbf{b}^\perp) = \widetilde{\mathbf{V}}_j^\top \mathbf{b}^\perp + \widetilde{v}_j \mathbf{b}^\perp \\
\mathbf{V}_j \mathbf{B} = \widetilde{\mathbf{V}}_j \mathbf{B} & \mathbf{V}_j \mathbf{a}^\perp = \widetilde{\mathbf{V}}_j \mathbf{a}^\perp + \widetilde{v}_j \mu^{-1} \mathbf{a}^\perp ({\mathbf{b}^\perp}^\top \mathbf{a}^\perp) = \widetilde{\mathbf{V}}_j \mathbf{a}^\perp + \widetilde{v}_j \mathbf{a}^\perp
\end{array}}$$

The challenge trapdoor is now $\mathsf{Tr}_0^* = (\mathsf{tr}_0^{[0]}, \mathsf{tr}_1^{[0]})$:

$$\begin{aligned}
\mathsf{tr}_0^{[0]} &= \left\llbracket \mathbf{Br} + \mathbf{a}^\perp \widehat{r} \right\rrbracket_2 \\
\mathsf{tr}_1^{[0]} &= \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}^*} + (\tilde{\omega}^* \cdot \widetilde{\mathbf{U}}_{\widetilde{\mathsf{S}}^*} + \widetilde{\mathbf{V}}_{\mathsf{R}^*})(\mathbf{Br} + \mathbf{a}^\perp \widehat{r}) + \boxed{(\tilde{\omega}^* \widetilde{u}_{\widetilde{\mathsf{S}}^*} + \widetilde{v}_{\mathsf{R}^*})}\, \mathbf{a}^\perp \widehat{r} \right\rrbracket_2
\end{aligned}$$

On the other hand, $\mathsf{Ct}_i = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j \in [\ell]})$ for $|\mathcal{R}| = \ell$ and a random permutation $\tau : [\ell] \to [\ell]$ where in both the games, $\mathsf{ct}_0 = \left\llbracket \mathbf{As} + \mathbf{b}^\perp \widehat{s} \right\rrbracket_1$ and $\kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top (\mathbf{As} + \mathbf{b}^\perp \widehat{s}) \right\rrbracket_{\mathrm{T}}$ for $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k, \widehat{s} \hookleftarrow \mathbb{Z}_p$. We now note $\mathsf{ct}_{1,j}$ of Ct_i in both games.

$$\mathsf{ct}_{1,j} = \begin{cases} \left\llbracket (\omega \cdot \mathbf{U}_{\mathsf{S}} + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)(\mathbf{As} + \mathbf{b}^\perp \widehat{s}) + \boxed{(\omega \widetilde{u}_{\mathsf{S}} + \widetilde{v}_{\mathsf{R}_{\tau(j)}})}\, \mathbf{b}^\perp \widehat{s} \right\rrbracket_1 & \text{in } \mathsf{Game}_{2,i,1} \\ \left\llbracket \mathbf{b}^\perp \widehat{u}_{i,j} + (\omega \cdot \mathbf{U}_{\mathsf{S}} + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)(\mathbf{As} + \mathbf{b}^\perp \widehat{s}) + \boxed{(\omega \widetilde{u}_{\mathsf{S}} + \widetilde{v}_{\mathsf{R}_{\tau(j)}})}\, \mathbf{b}^\perp \widehat{s} \right\rrbracket_1 & \text{in } \mathsf{Game}_{2,i,2} \end{cases}$$

We focus on the boxed-parts of $(\mathsf{ct}_{1,j})_{j\in[\ell]}$ and $\mathsf{tr}_1^{[0]}$ above. We argue that $\left\{\sigma_j = (\omega\widetilde{u}_{\mathsf{S}} + \widetilde{v}_{\mathsf{R}_{\tau(j)}})\right\}_{j\in[\ell]}$ and $\sigma_{\ell+1} = (\tilde{\omega}^*\widetilde{u}_{\widetilde{\mathsf{S}}^*} + \widetilde{v}_{\mathsf{R}^*})$ are uniformly random and independently distributed. This ensures that $\mathbf{b}^{\perp}\widehat{u}_{i,j}$ are hidden in $\mathsf{ct}_{1,j}$ of $\mathsf{Game}_{2,i,2}$ ensuring type-1 and type-2 semi-functional ciphertexts are indistinguishable even given the challenge semi-functional trapdoor. We show this independence next while expressing $(\sigma_i)_{i\in[\ell+1]}$ as a system of linear equations:

Case-1 Consider the case $\widetilde{\mathsf{S}}^* \neq \mathsf{S}$. (Wlog we also consider $\tilde{\omega}^* = \omega$ and $\widetilde{v}_{\mathsf{R}^*} = \widetilde{v}_{\mathsf{R}_{\tau(j)}}$ for some $j \in [\ell]$.)

$$\begin{bmatrix}\sigma_1\\ \sigma_2\\ \vdots\\ \sigma_{\ell+1}\end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix}\omega & 0 & 1 & 0 & \cdots & 0 & \cdots & 0\\ \omega & 0 & 0 & 1 & \cdots & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots\\ \omega & 0 & 0 & 0 & \cdots & 1 & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots\\ \omega & 0 & 0 & 0 & \cdots & 0 & \cdots & 1\\ 0 & \tilde{\omega}^* & 0 & 0 & \cdots & \underset{\tau(j)^{\text{th}}}{\underset{\uparrow}{1}} & \cdots & 0\end{bmatrix}_{(\ell+1)\times(\ell+2)} \times \begin{bmatrix}\widetilde{u}_{\mathsf{S}}\\ \widetilde{u}_{\widetilde{\mathsf{S}}^*}\\ \widetilde{v}_{\mathsf{R}_{\tau(1)}}\\ \widetilde{v}_{\mathsf{R}_{\tau(2)}}\\ \vdots\\ \widetilde{v}_{\mathsf{R}_{\tau(j)}}\\ \vdots\\ \widetilde{v}_{\mathsf{R}_{\tau(\ell)}}\end{bmatrix}_{(\ell+2)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell + 1)$. Since, $\widetilde{u}_{\mathsf{S}}, \widetilde{u}_{\widetilde{\mathsf{S}}^*}, \widetilde{v}_{\mathsf{R}_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}_{\tau(\ell)}}$ are sampled uniformly at random, $\sigma_1, \ldots, \sigma_{\ell+1}$ are too independent and uniformly random.

Case-2 Consider the case $\widetilde{\mathsf{S}}^* = \mathsf{S} \wedge \tilde{\omega}^* \neq \omega$. Wlog we consider $\widetilde{v}_{\mathsf{R}^*} = \widetilde{v}_{\mathsf{R}_{\tau(j)}}$ for some $j \in [\ell]$.

$$\begin{bmatrix}\sigma_1\\ \sigma_2\\ \vdots\\ \sigma_{\ell+1}\end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix}\omega & 1 & 0 & \cdots & 0 & \cdots & 0\\ \omega & 0 & 1 & \cdots & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots\\ \omega & 0 & 0 & \cdots & 1 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots\\ \omega & 0 & 0 & \cdots & 0 & \cdots & 1\\ \tilde{\omega}^* & 0 & 0 & \cdots & \underset{\tau(j)^{\text{th}}}{\underset{\uparrow}{1}} & \cdots & 0\end{bmatrix}_{(\ell+1)\times(\ell+1)} \times \begin{bmatrix}\widetilde{u}_{\mathsf{S}}\\ \widetilde{v}_{\mathsf{R}_{\tau(1)}}\\ \widetilde{v}_{\mathsf{R}_{\tau(2)}}\\ \vdots\\ \widetilde{v}_{\mathsf{R}_{\tau(j)}}\\ \vdots\\ \widetilde{v}_{\mathsf{R}_{\tau(\ell)}}\end{bmatrix}_{(\ell+1)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell + 1)$. Since, $\widetilde{u}_{\mathsf{S}}, \widetilde{v}_{\mathsf{R}_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}_{\tau(\ell)}}$ are sampled uniformly at random and $\tilde{\omega}^* \neq \omega$, $\sigma_1, \ldots, \sigma_{\ell+1}$ are also independent and uniformly random.

Case-3 Consider the case $\widetilde{\mathsf{S}}^* = \mathsf{S} \wedge \tilde{\omega}^* = \omega \wedge \mathsf{R}^* \notin \mathcal{R}$. So, we consider $\widetilde{v}_{\mathsf{R}^*} \neq \widetilde{v}_{\mathsf{R}_{\tau(j)}}$ for all $j \in [\ell]$.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{\ell+1} \end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix} \omega\ 1\ 0 \cdots & 0 & \cdots 0\ 0 \\ \omega\ 0\ 1 \cdots & 0 & \cdots 0\ 0 \\ \vdots\ \vdots\ \vdots\ \ddots & \vdots & \ddots\ \vdots\ \vdots \\ \omega\ 0\ 0 \cdots & 1 & \cdots 0\ 0 \\ \vdots\ \vdots\ \vdots\ \ddots & \vdots & \ddots\ \vdots\ \vdots \\ \omega\ 0\ 0 \cdots & 0 & \cdots 1\ 0 \\ \omega\ 0\ 0 \cdots & 0 & \cdots 0\ 1 \\ & \overset{\uparrow}{\tau(j)^{\text{th}}} & \end{bmatrix}_{(\ell+1)\times(\ell+2)} \times \begin{bmatrix} \widetilde{u}_{\mathsf{S}} \\ \widetilde{v}_{\mathsf{R}_{\tau(1)}} \\ \widetilde{v}_{\mathsf{R}_{\tau(2)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}_{\tau(j)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}_{\tau(\ell)}} \\ \widetilde{v}_{\mathsf{R}^*} \end{bmatrix}_{(\ell+2)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell + 1)$. Since, $\widetilde{u}_{\mathsf{S}}, \widetilde{v}_{\mathsf{R}_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}_{\tau(\ell)}}, \widetilde{v}_{\mathsf{R}^*}$ are sampled uniformly at random, $\sigma_1, \ldots, \sigma_{\ell+1}$ are also independent and uniformly random.

This naturally ensures that $\mathbf{b}^{\perp}\widehat{u}_{i,j}$ are hidden in $\mathsf{ct}_{1,j}$ of $\mathsf{Game}_{2,i,2}$. Thus, $\mathsf{Game}_{2,i,1}$ and $\mathsf{Game}_{2,i,2}$ are identically distributed. □

Proof of Lemma 4. Same as Proof of Lemma 2. □

Proof of Lemma 5. We make a conceptual change here.

- **Setup**: We sample $\mathfrak{b} \hookleftarrow \{0,1\}$, $\mathbf{A}, \mathbf{B} \hookleftarrow \mathcal{D}_k$ and $\mathbf{a}^{\perp}, \mathbf{b}^{\perp} \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^{\perp}}^{\top}\mathbf{A} = {\mathbf{b}^{\perp}}^{\top}\mathbf{B} = \mathbf{0}$. We publish $\mathsf{pp} = ([\![\mathbf{A}]\!]_1, [\![\mathbf{A}]\!]_2, [\![\mathbf{B}]\!]_1, [\![\mathbf{B}]\!]_2)$.
- **Query Phase-I**:
  - $O_{\mathsf{pk}}(j)$: We sample $\boldsymbol{\alpha}_j \hookleftarrow \mathbb{Z}_p^{(k+1)}$, $\mathbf{U}_j, \mathbf{V}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$ to set $\mathsf{sk}_j = ([\![\boldsymbol{\alpha}_j]\!]_2, \left[\!\!\left[\mathbf{U}_j^{\top}\mathbf{A}\right]\!\!\right]_1, [\![\mathbf{V}_j\mathbf{B}]\!]_2)$ and publish $\mathsf{pk}_j = (\left[\!\!\left[\boldsymbol{\alpha}_j^{\top}\mathbf{A}\right]\!\!\right]_T, [\![\mathbf{U}_j\mathbf{B}]\!]_2, \left[\!\!\left[\mathbf{V}_j^{\top}\mathbf{A}\right]\!\!\right]_1)$.
  - $O_{\mathsf{Tr}}(\mathsf{pk}_{\widetilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{pk}_{\mathsf{R}})$: We sample $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$ and reply with trapdoor $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$, where
$$\mathsf{tr}_0 = [\![\mathbf{Br}]\!]_2, \mathsf{tr}_1 = \left[\!\!\left[\boldsymbol{\alpha}_{\mathsf{R}} + (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{Br}\right]\!\!\right]_2.$$
  - $O_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$: On i^{th} query, we sample $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$ and $\mathbf{u}_{i,j} \hookleftarrow \mathbb{Z}_p^{k+1}$ for $j \in [|\mathcal{R}|]$. We output $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[|\mathcal{R}|]})$ where,
$$\mathsf{ct}_0 = [\![\mathbf{As}]\!]_1, \mathsf{ct}_{1,j} = \left[\!\!\left[\mathbf{u}_{i,j} + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^{\top}\mathbf{As}\right]\!\!\right]_1, \kappa = \left[\!\!\left[\boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^{\top}\mathbf{As}\right]\!\!\right]_T.$$
- **Challenge**: Given the challenge $(\mathsf{pk}_{\widetilde{\mathsf{S}}^*}, \tilde{\omega}^*, \mathsf{pk}_{\mathsf{R}^*})$, we sample $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$, $\mathbf{z} \hookleftarrow \mathbb{Z}_p^{k+1}$ and $\widehat{r} \in \mathbb{Z}_p$. Then we return $\mathsf{Tr}_0^* = (\mathsf{tr}_0^{[0]}, \mathsf{tr}_1^{[0]})$ where,
$$\mathsf{tr}_0 = \left[\!\!\left[\mathbf{Br} + \mathbf{a}^{\perp}\widehat{r}\right]\!\!\right]_2, \mathsf{tr}_1 = \left[\!\!\left[\boldsymbol{\alpha}_{\mathsf{R}^*} + \mathbf{z} + \mathbf{V}_{\mathsf{R}^*}(\mathbf{Br} + \mathbf{a}^{\perp}\widehat{r})\right]\!\!\right]_2.$$

– **Query Phase-**II: Same as **Query Phase-**I.

We first argue that the modified ciphertexts are distributed the same way as in $\mathsf{Game}_{2,Q,3}$. For that, we argue that $\mathbf{U}_{\mathsf{S}}^\top\mathbf{A}\mathbf{s}+\mathbf{b}^\perp\widehat{u}_{i,j}$ is uniformly random. To see this, we left multiply them with $\begin{pmatrix}\mathbf{B}^\top\\ {\mathbf{a}^\perp}^\top\end{pmatrix}$. Then,

$$\begin{pmatrix}\mathbf{B}^\top\\ {\mathbf{a}^\perp}^\top\end{pmatrix}\cdot(\mathbf{U}_{\mathsf{S}}^\top\mathbf{A}\mathbf{s}+\mathbf{b}^\perp\widehat{u}_{i,j}) = \begin{pmatrix}\mathbf{B}^\top\mathbf{U}_{\mathsf{S}}^\top\mathbf{A}\mathbf{s}\\ {\mathbf{a}^\perp}^\top\mathbf{U}_{\mathsf{S}}^\top\mathbf{A}\mathbf{s}+{\mathbf{a}^\perp}^\top\mathbf{b}^\perp\widehat{u}_{i,j}\end{pmatrix}$$

such that ${\mathbf{a}^\perp}^\top\mathbf{b}^\perp\widehat{u}_{i,j}$ is an uniformly random value. As $\begin{pmatrix}\mathbf{B}^\top\\ {\mathbf{a}^\perp}^\top\end{pmatrix}$ spans the space, $\mathbf{U}_{\mathsf{S}}^\top\mathbf{A}\mathbf{s}+{\mathbf{b}^\perp}^\top\widehat{u}_{i,j}$ is uniformly random. Therefore replacing this with uniformly random $\mathbf{u}_{i,j}$ is invisible to $\mathcal{A}$.

Now we argue that, $\mathbf{U}_{\widetilde{\mathsf{S}}^*}(\mathbf{B}\mathbf{r}+\mathbf{a}^\perp\widehat{r})$ in $\mathsf{tr}_1^{[0]}$ of $\mathsf{Game}_{2,Q,3}$ is uniformly random. Observe that two mutually exclusive situations can happen that is exhaustive.

1. No ciphertext query involved $\widetilde{\mathsf{S}}^*$: $\mathbf{U}_{\widetilde{\mathsf{S}}^*}$ is completely hidden from $\mathcal{A}$.
2. If $\widetilde{\mathsf{S}}^*$ was used for ciphertext queries: $\mathbf{U}_{\widetilde{\mathsf{S}}^*}^\top\mathbf{A}$ is completely hidden in the ciphertexts by above argument.

Thus, $\mathcal{A}$ only gets $\mathbf{U}_{\widetilde{\mathsf{S}}^*}\mathbf{B}$ from $\mathsf{pk}_{\widetilde{\mathsf{S}}^*}$ and any trapdoor query made on $\widetilde{\mathsf{S}}^*$. As $\mathbf{a}^\perp$ is outside the span of $\mathbf{B}$, $\mathbf{U}_{\widetilde{\mathsf{S}}^*}(\mathbf{B}\mathbf{r}+\mathbf{a}^\perp\widehat{r})$ is independent from $\mathsf{pk}_{\widetilde{\mathsf{S}}^*}$. As $\mathbf{U}_{\widetilde{\mathsf{S}}^*}(\mathbf{B}\mathbf{r}+\mathbf{a}^\perp\widehat{r})$ is independent from $\mathbf{U}_{\widetilde{\mathsf{S}}^*}\mathbf{B}$, we replace it by $\mathbf{z}\hookleftarrow\mathbb{Z}_p^{k+1}$. □

Proof of Lemma 6. Finally, we state that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_3}(\lambda)=0$. This is because, $\mathbf{z}$ is a uniformly random vector that hides $\mathbf{V}_{\mathsf{R}^*}(\mathbf{B}\mathbf{r}+\mathbf{a}^\perp\widehat{r})$ in $\mathsf{tr}_1^{[0]}$. □

4.2 Ciphertext Security

Proof of Theorem 2. The proof is done via a hybrid argument similar to the dual system proof technique [22]. In this proof technique, both the challenge ciphertexts and the queried trapdoors can be of two kinds: (i) normal and (ii) semi-functional. Note that a semi-functional trapdoor cannot *match* a semi-functional ciphertext even if they encode *matching* attributes.

We informally describe the sequence of games here. Game_0 is the real security game where the challenge ciphertext, all the queried trapdoors, and ciphertexts are normal. The challenge ciphertext is made semi-functional in Game_1. Following subsequence of games $(\mathsf{Game}_{2,i,1},\mathsf{Game}_{2,i,2},\mathsf{Game}_{2,i,3})_{i\in[q]}$ make the queried trapdoors semi-functional. In particular, in $\mathsf{Game}_{2,i,j}$ the i^{th} trapdoor is type-j semi-functional. In Game_3, we replace the challenge $(\kappa_j^{[0]})_{j\in[|\mathcal{R}|]}$ by independent random choices. Finally, in Game_4, we replace the challenge $(\mathsf{ct}_{1,j}^{[0]})_{j\in[|\mathcal{R}|]}$ by independent random choices. Note that, we denote Game_1 by $\mathsf{Game}_{2,0}$ as well. The indistinguishability of Game_0 and Game_4 is proven via the sequence of games mentioned in Table 2.

Table 2. Outline of the proof strategy

Games	Difference from	Indistinguishability	
	Previous Game	from Previous Game	under Assumption
Game_0	-	-	
Game_1	challenge ciphertext Ct_0^* is semi-functional	Lemma 7	*bil*-$\mathcal{D}_k$-matDH
$\mathsf{Game}_{2,i,1}$	i^{th} trapdoor is type-1 semi-functional	Lemma 8	*bil*-$\mathcal{D}_k$-matDH
$\mathsf{Game}_{2,i,2}$	i^{th} trapdoor is type-2 semi-functional	Lemma 9	information theoretic
$\mathsf{Game}_{2,i,3}$	i^{th} trapdoor is type-3 semi-functional	Lemma 10	*bil*-$\mathcal{D}_k$-matDH
Game_3	challenge $(\kappa_j^{[0]})_{j\in[\lvert\mathcal{R}\rvert]}$ are random	Lemma 11	information theoretic
Game_4	challenge $(\mathsf{ct}_{1,j}^{[0]})_{j\in[\lvert\mathcal{R}\rvert]}$ are random	Lemma 12	information theoretic

Similar to the proof of Theorem 1, we argue the indistinguishability of Game_0 and Game_4 via a sequence of games overviewed in Table 2. Next, we give the semi-functional algorithms we would need in the proof.

4.2.1 Semi-functional Algorithms

- $\mathsf{sfSrchEnc}(\mathsf{pp}, \mathsf{sk}_\mathsf{S}, \omega, \mathsf{sk}_\mathcal{R})$: Let $\mathsf{sk}_\mathcal{R} = \{\mathsf{sk}_{\mathsf{R}_1}, \ldots, \mathsf{sk}_{\mathsf{R}_\ell}\}$. Choose $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$, $\widehat{s} \hookleftarrow \mathbb{Z}_p$ and $\mathbf{b}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{b}^\perp}^\top \mathbf{B} = 0$. Compute the semi-functional ciphertext $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$ as follows:
  - $\mathsf{ct}_0 = \left[\!\left[\mathbf{As} + \mathbf{b}^\perp \widehat{s}\right]\!\right]_1$,
  - $\mathsf{ct}_{1,j} = \left[\!\left[(\omega \cdot \mathbf{U}_\mathsf{S}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)(\mathbf{As} + \mathbf{b}^\perp \widehat{s})\right]\!\right]_1$,
  - $\kappa_j = \left[\!\left[\boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top (\mathbf{As} + \mathbf{b}^\perp \widehat{s})\right]\!\right]_\mathrm{T}$.
- $\mathsf{sfTrapdoor}(\mathsf{pp}, \mathsf{sk}_{\widetilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{sk}_\mathsf{R})$: Choose $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$, $\delta_\mathsf{R}, \widehat{v}, \widehat{r} \hookleftarrow \mathbb{Z}_p$ and $\mathbf{a}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^\perp}^\top \mathbf{A} = 0$. Compute the semi-functional trapdoors as $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$ as follows.

$$(\mathsf{tr}_0, \mathsf{tr}_1) = \left(\left[\!\left[\mathbf{Br} + \mu \mathbf{a}^\perp \widehat{r}\right]\!\right]_2, \left[\!\left[\boldsymbol{\alpha}_\mathsf{R} + \nu \mathbf{a}^\perp \delta_\mathsf{R} + \nu \mathbf{a}^\perp \widehat{v} + (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_\mathsf{R})(\mathbf{Br} + \mu \mathbf{a}^\perp \widehat{r})\right]\!\right]_2\right).$$

  where if $(\nu = 0, \mu = 1)$, Tr is a type-1 semi-functional trapdoor; if $(\nu = 1, \mu = 1)$, Tr is a type-2 semi-functional trapdoor; and if $(\nu = 1, \mu = 0)$, Tr is a type-3 semi-functional trapdoor.

4.2.2 Sequence of Games

Lemma 7. (Game_0 *to* Game_1) *For any* ppt *adversary* $\mathcal{A}$ *that makes at most* q *trapdoor queries, at most* Q *encryption requests, and at most* $\tilde{Q}$ *public key requests; there exists a* ppt *adversary* $\mathcal{B}$ *such that,*

$$|\mathsf{Adv}_\mathcal{A}^{\mathsf{Game}_0}(\lambda) - \mathsf{Adv}_\mathcal{A}^{\mathsf{Game}_1}(\lambda)| \le \mathsf{Adv}_\mathcal{B}^{bil\text{-}\mathcal{D}_k\text{-}\mathsf{matDH}}.$$

Lemma 8. ($\mathsf{Game}_{2,i-1,3}$ *to* $\mathsf{Game}_{2,i,1}$) *For any* ppt *adversary* $\mathcal{A}$ *that makes at most* q *trapdoor queries, at most* Q *encryption requests, and at most* $\tilde{Q}$ *public key requests; there exists a* ppt *adversary* $\mathcal{B}$ *such that,*

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i-1,3}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,1}}(\lambda)| \leq \mathsf{Adv}_{\mathcal{B}}^{bil\text{-}\mathcal{D}_k\text{-}\mathsf{matDH}}.$$

Lemma 9. ($\mathsf{Game}_{2,i,1}$ *to* $\mathsf{Game}_{2,i,2}$) *For any adversary* $\mathcal{A}$,

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,1}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,2}}(\lambda)| = 0.$$

Lemma 10. ($\mathsf{Game}_{2,i,2}$ *to* $\mathsf{Game}_{2,i,3}$) *For any* ppt *adversary* $\mathcal{A}$ *that makes at most* q *trapdoor queries, at most* Q *encryption requests, and at most* $\tilde{Q}$ *public key requests; there exists a* ppt *adversary* $\mathcal{B}$ *such that,*

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,2}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,i,3}}(\lambda)| \leq \mathsf{Adv}_{\mathcal{B}}^{bil\text{-}\mathcal{D}_k\text{-}\mathsf{matDH}}.$$

Lemma 11. ($\mathsf{Game}_{2,q,3}$ *to* Game_{3}) *For any adversary* $\mathcal{A}$,

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2,q,3}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{3}}(\lambda)| = 0.$$

Lemma 12. (Game_{3} *to* Game_{4}) *For any adversary* $\mathcal{A}$,

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{3}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{4}}(\lambda)| = 0.$$

Lemma 13. (Game_{4}) *For any adversary* $\mathcal{A}$, $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{4}}(\lambda)| = 0$.

4.2.3 Proof of Games

Proof of Lemma 7. $\mathcal{B}$ receives a *bil*-$\mathcal{D}_k$-matDH problem instance ($[\![\mathbf{M}]\!]_1$, $[\![\mathbf{z}]\!]_1$, $[\![\mathbf{M}]\!]_2$, $[\![\mathbf{z}]\!]_2$), where $\mathbf{z} = \mathbf{M}\mathbf{y}$ or $\mathbf{z} \hookleftarrow \mathbb{Z}_p^{k+1}$.

- **Setup**: $\mathcal{B}$ samples $\mathfrak{b} \hookleftarrow \{0,1\}$, $\mathbf{B} \leftarrow \mathcal{D}_k$ and $\mathbf{b}^{\perp} \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{b}^{\perp}}^{\top}\mathbf{B} = \mathbf{0}$. It publishes $\mathsf{pp} = ([\![\mathbf{M}]\!]_1, [\![\mathbf{M}]\!]_2, [\![\mathbf{B}]\!]_1, [\![\mathbf{B}]\!]_2)$.
- **Query Phase-**I: $\mathcal{A}$ makes following queries–
  - $O_{\mathsf{pk}}(j)$: $\mathcal{B}$ samples $\boldsymbol{\alpha}_j \hookleftarrow \mathbb{Z}_p^{k+1}$, $\mathbf{U}_j, \mathbf{V}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$. It returns $\mathsf{pk}_j = ([\![\boldsymbol{\alpha}_j^{\top}]\!]_{\mathrm{T}}, [\![\mathbf{U}_j\mathbf{B}]\!]_2, \left[\!\!\left[\mathbf{V}_j^{\top}\mathbf{M}\right]\!\!\right]_1)$.
  - $O_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$: Let $\mathsf{pk}_{\mathcal{R}} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$. $\mathcal{B}$ samples a random permutation $\tau : [\ell] \to [\ell]$ and $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$ to output $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$ where $\mathsf{ct}_0 = [\![\mathbf{Ms}]\!]_1$, $\mathsf{ct}_{1,j} = \left[\!\!\left[(\omega \cdot \mathbf{U}_{\mathsf{S}}^{\top} + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^{\top})\mathbf{Ms}\right]\!\!\right]_1$, $\kappa_j = \left[\!\!\left[\boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^{\top}\mathbf{Ms}\right]\!\!\right]_{\mathrm{T}}$.
  - $O_{\mathsf{Tr}}(\mathsf{pk}_{\tilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{pk}_{\mathsf{R}})$: $\mathcal{B}$ samples $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$. It outputs $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$, where $\mathsf{tr}_0 = [\![\mathbf{Br}]\!]_2$, $\mathsf{tr}_1 = [\![\boldsymbol{\alpha}_{\mathsf{R}} + (\tilde{\omega} \cdot \mathbf{U}_{\tilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{Br}]\!]_2$.
- **Challenge**: $\mathcal{A}$ makes a challenge on $(\mathsf{pk}_{\mathsf{S}^*}, \omega^*, \mathsf{pk}_{\mathcal{R}^*})$ where $\mathsf{pk}_{\mathcal{R}^*} = \{\mathsf{pk}_{\mathsf{R}_1^*}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell^*}\}$. $\mathcal{B}$ defines a random permutation $\tau : [\ell] \to [\ell]$ to compute $\mathsf{Ct}_0^* = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}^{[0]}, \kappa_j^{[0]})_{j\in[\ell]})$ where

$$\mathsf{ct}_0 = [\![\mathbf{z}]\!]_1, \mathsf{ct}_{1,j}^{[0]} = \left[\!\!\left[(\omega^* \cdot \mathbf{U}_{\mathsf{S}^*}^{\top} + \mathbf{V}_{\mathsf{R}_{\tau(j)}^*}^{\top})\mathbf{z}\right]\!\!\right]_1, \kappa_j^{[0]} = \left[\!\!\left[\boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}^*}^{\top}\mathbf{z}\right]\!\!\right]_{\mathrm{T}}.$$

  $\mathcal{B}$ also samples $\mathsf{Ct}_1^* \hookleftarrow \mathcal{CT}$ and returns $\mathsf{Ct}_{\mathfrak{b}}^*$ to $\mathcal{A}$.

- **Query Phase-II**: Same as **Query Phase-I**.
- **Guess**: $\mathcal{B}$ forwards $\mathfrak{b}'$ that is output by $\mathcal{A}$.

It is easy to see that if $\mathbf{z} = \mathbf{M}\mathbf{y}$, Ct_0^* is normal; and if $\mathbf{z} \hookleftarrow \mathbb{Z}_p^{k+1}$, Ct_0^* is semi-functional. □

Proof of Lemma 8. $\mathcal{B}$ receives a *bil-*$\mathcal{D}_k$-matDH problem instance $(\llbracket \mathbf{N} \rrbracket_1, \llbracket \mathbf{y} \rrbracket_1, \llbracket \mathbf{N} \rrbracket_2, \llbracket \mathbf{y} \rrbracket_2)$, where $\mathbf{y} = \mathbf{N}\mathbf{z}$ or $\mathbf{y} \hookleftarrow \mathbb{Z}_p^{k+1}$.

- **Setup**: $\mathcal{B}$ samples $\mathfrak{b} \hookleftarrow \{0,1\}$, $\mathbf{A} \leftarrow \mathcal{D}_k$ and $\mathbf{a}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^\perp}^\top \mathbf{A} = \mathbf{0}$. It publishes $\mathsf{pp} = (\llbracket \mathbf{A} \rrbracket_1, \llbracket \mathbf{A} \rrbracket_2, \llbracket \mathbf{N} \rrbracket_1, \llbracket \mathbf{N} \rrbracket_2)$.
- **Query Phase-I**: $\mathcal{A}$ makes following queries–
  - $O_{\mathsf{pk}}(j)$: $\mathcal{B}$ samples $\boldsymbol{\alpha}_j \hookleftarrow \mathbb{Z}_p^{k+1}$, $\mathbf{U}_j, \mathbf{V}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$. It returns $\mathsf{pk}_j = (\llbracket \boldsymbol{\alpha}_j^\top \rrbracket_{\mathrm{T}}, \llbracket \mathbf{U}_j \mathbf{N} \rrbracket_2, \llbracket \mathbf{V}_j^\top \mathbf{A} \rrbracket_1)$.
  - $O_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$: Let $\mathsf{pk}_{\mathcal{R}} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$. $\mathcal{B}$ samples a random permutation $\tau : [\ell] \to [\ell]$ and $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$. It outputs $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$ where
$$\mathsf{ct}_0 = \llbracket \mathbf{A}\mathbf{s} \rrbracket_1, \mathsf{ct}_{1,j} = \left\llbracket (\omega \cdot \mathbf{U}_{\mathsf{S}}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^\top)\mathbf{A}\mathbf{s} \right\rrbracket_1, \kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^\top \mathbf{A}\mathbf{s} \right\rrbracket_{\mathrm{T}}.$$
  - $O_{\mathsf{Tr}}(\mathsf{pk}_{\widetilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{pk}_{\mathsf{R}})$: On t^{th} trapdoor query, $\mathcal{B}$ does the following:
    - $-t > i$**:** $\mathcal{B}$ samples $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$. It outputs $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$, where $\mathsf{tr}_0 = \llbracket \mathbf{N}\mathbf{r} \rrbracket_2, \mathsf{tr}_1 = \llbracket \boldsymbol{\alpha}_{\mathsf{R}} + (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{N}\mathbf{r} \rrbracket_2$.
    - $-t < i$**:** It samples $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$ and $\delta_{\mathsf{R}}, \widehat{v}_t \hookleftarrow \mathbb{Z}_p$ to output $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$, where
$$\mathsf{tr}_0 = \llbracket \mathbf{N}\mathbf{r} \rrbracket_2, \mathsf{tr}_1 = \llbracket \boldsymbol{\alpha}_{\mathsf{R}} + \mathbf{a}^\perp \delta_{\mathsf{R}} + \mathbf{a}^\perp \widehat{v}_t + (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{N}\mathbf{r} \rrbracket_2.$$
    - $-t = i$**:** It outputs $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$ where
$$\mathsf{tr}_0 = \llbracket \mathbf{y} \rrbracket_2, \mathsf{tr}_1 = \llbracket \boldsymbol{\alpha}_{\mathsf{R}} + (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{y} \rrbracket_2.$$
- **Challenge**: $\mathcal{A}$ makes a challenge on $(\mathsf{pk}_{\mathsf{S}^*}, \omega^*, \mathsf{pk}_{\mathcal{R}^*})$ where $\mathsf{pk}_{\mathcal{R}^*} = \{\mathsf{pk}_{\mathsf{R}_1^*}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell^*}\}$. $\mathcal{B}$ samples a random permutation $\tau : [\ell] \to [\ell]$ and $\widetilde{\mathbf{s}} \hookleftarrow \mathbb{Z}_p^{k+1}$. It then computes $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$ where
$$\mathsf{ct}_0 = \llbracket \widetilde{\mathbf{s}} \rrbracket_1, \mathsf{ct}_{1,j} = \left\llbracket (\omega^* \cdot \mathbf{U}_{\mathsf{S}^*}^\top + \mathbf{V}_{\mathsf{R}_{\tau(j)}^*}^\top)\widetilde{\mathbf{s}} \right\rrbracket_1, \kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}^*}^\top \widetilde{\mathbf{s}} \right\rrbracket_{\mathrm{T}}.$$
  $\mathcal{B}$ also samples $\mathsf{Ct}_1^* \hookleftarrow \mathcal{CT}$ and returns $\mathsf{Ct}_{\mathfrak{b}}^*$ to $\mathcal{A}$.
- **Query Phase-II**: Same as **Query Phase-I**.
- **Guess**: $\mathcal{B}$ forwards $\mathfrak{b}'$ that is output by $\mathcal{A}$.

It is easy to see that if $\mathbf{y} = \mathbf{N}\mathbf{z}$, the i^{th} trapdoor response Tr is normal; and if $\mathbf{y} \hookleftarrow \mathbb{Z}_p^{k+1}$, the i^{th} trapdoor response Tr is semi-functional. □

Proof of Lemma 9. Recall that, the challenge ciphertext Ct_0^* encodes $(\mathsf{S}^*, \omega^*, \mathcal{R}^*)$ and the i^{th} trapdoor Tr_i encodes $(\widetilde{\mathsf{S}}, \tilde{\omega}, \mathsf{R})$. We here argue that, the joint distributions of $\{\mathsf{pp}, \mathsf{PK}, \mathsf{Tr}_i, \mathsf{Ct}_0^*\}$ if Tr_i is a type-1 semi-functional trapdoor *is identical to* the joint distributions of $\{\mathsf{pp}, \mathsf{PK}, \mathsf{Tr}_i, \mathsf{Ct}_0^*\}$ if Tr_i is a type-2 semi-functional trapdoor where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{PK} = \{\mathsf{pk}_j\}_{j \in \mathcal{R}^* \cup \{\mathsf{S}^*, \mathsf{R}, \widetilde{\mathsf{S}}\}}$ for $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, j)$ and $\mathsf{Ct}_0^* = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}^{[0]}, \kappa_j^{[0]})_{j \in [\ell]}) \leftarrow \mathsf{SFEncrypt}(\mathsf{sk}_{\mathsf{S}^*}, \omega^*, \mathsf{sk}_{\mathcal{R}^*})$ is a semi-functional ciphertext for $\mathsf{sk}_{\mathcal{R}^*} = \{\mathsf{sk}_{\mathsf{R}_1^*}, \dots, \mathsf{sk}_{\mathsf{R}_\ell^*}\}$.

We sample $\mathbf{A}, \mathbf{B} \leftarrow \mathcal{D}_k$ and $\mathbf{a}^\perp, \mathbf{b}^\perp \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^\perp}^\top \mathbf{A} = {\mathbf{b}^\perp}^\top \mathbf{B} = 0$ but $\eta = {\mathbf{a}^\perp}^\top \mathbf{b}^\perp \neq 0$. Now we sample $\mathbf{U}_j, \mathbf{V}_j$ in a different way for any user j. In particular, we define $\mathbf{U}_j = \widetilde{\mathbf{U}}_j + \widetilde{u}_j \eta^{-1} \mathbf{a}^\perp {\mathbf{b}^\perp}^\top$ and $\mathbf{V}_j = \widetilde{\mathbf{V}}_j + \widetilde{v}_j \eta^{-1} \mathbf{a}^\perp {\mathbf{b}^\perp}^\top$ for $\widetilde{\mathbf{U}}_j, \widetilde{\mathbf{V}}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$ and $\widetilde{u}_j, \widetilde{v}_j \hookleftarrow \mathbb{Z}_p$. Observe from the following equations pk_j stays the same for all user j.

$$\boxed{\begin{array}{ll}
\mathbf{U}_j^\top \mathbf{A} = \widetilde{\mathbf{U}}_j^\top \mathbf{A} & \mathbf{U}_j^\top \mathbf{b}^\perp = \widetilde{\mathbf{U}}_j^\top \mathbf{b}^\perp + \widetilde{u}_j \eta^{-1} \mathbf{b}^\perp ({\mathbf{a}^\perp}^\top \mathbf{b}^\perp) = \widetilde{\mathbf{U}}_j^\top \mathbf{b}^\perp + \widetilde{u}_j \mathbf{b}^\perp \\
\mathbf{U}_j \mathbf{B} = \widetilde{\mathbf{U}}_j \mathbf{B} & \mathbf{U}_j \mathbf{a}^\perp = \widetilde{\mathbf{U}}_j \mathbf{a}^\perp + \widetilde{u}_j \eta^{-1} \mathbf{a}^\perp ({\mathbf{b}^\perp}^\top \mathbf{a}^\perp) = \widetilde{\mathbf{U}}_j \mathbf{b}^\perp + \widetilde{u}_j \mathbf{a}^\perp \\
\mathbf{V}_j^\top \mathbf{A} = \widetilde{\mathbf{V}}_j^\top \mathbf{A} & \mathbf{V}_j^\top \mathbf{b}^\perp = \widetilde{\mathbf{V}}_j^\top \mathbf{b}^\perp + \widetilde{v}_j \eta^{-1} \mathbf{b}^\perp ({\mathbf{a}^\perp}^\top \mathbf{b}^\perp) = \widetilde{\mathbf{V}}_j^\top \mathbf{b}^\perp + \widetilde{v}_j \mathbf{b}^\perp \\
\mathbf{V}_j \mathbf{B} = \widetilde{\mathbf{V}}_j \mathbf{B} & \mathbf{V}_j \mathbf{a}^\perp = \widetilde{\mathbf{V}}_j \mathbf{a}^\perp + \widetilde{v}_j \eta^{-1} \mathbf{a}^\perp ({\mathbf{b}^\perp}^\top \mathbf{a}^\perp) = \widetilde{\mathbf{V}}_j \mathbf{a}^\perp + \widetilde{v}_j \mathbf{a}^\perp
\end{array}}$$

The challenge ciphertext is $\mathsf{Ct}_0^* = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}^{[0]}, \kappa_j^{[0]})_{j \in [\ell]})$ for $|\mathcal{R}| = \ell$ and the random permutation $\tau : [\ell] \to [\ell]$ such that $\mathsf{ct}_0 = \llbracket \mathbf{A}\mathbf{s} + \mathbf{b}^\perp \widehat{s} \rrbracket_1$,

$$\begin{aligned}
\mathsf{ct}_{1,j}^{[0]} &= \left\llbracket (\omega^* \cdot \widetilde{\mathbf{U}}_{\mathsf{S}^*}^\top + \widetilde{\mathbf{V}}_{\mathsf{R}^*_{\tau(j)}}^\top)(\mathbf{A}\mathbf{s} + \mathbf{b}^\perp \widehat{s}) + \boxed{(\omega^* \widetilde{u}_{\mathsf{S}^*} + \widetilde{v}_{\mathsf{R}^*_{\tau(j)}}) \mathbf{b}^\perp \widehat{s}} \right\rrbracket_2 \\
\kappa_j^{[0]} &= \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}^*_{\tau(j)}}^\top (\mathbf{A}\mathbf{s} + \mathbf{b}^\perp \widehat{s}) \right\rrbracket_T
\end{aligned}$$

On the other hand, $\mathsf{Tr}_i = (\mathsf{tr}_0, \mathsf{tr}_1)$ where in both the games, $\mathsf{tr}_0 = \llbracket \mathbf{B}\mathbf{r} + \mathbf{a}^\perp \widehat{r} \rrbracket_1$ for $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$ and $\widehat{r}, \delta_\mathsf{R}, \widehat{v}_i \hookleftarrow \mathbb{Z}_p$. We now note tr_1 of Tr_i in the both the games down.

$$\mathsf{tr}_1 = \begin{cases}
\left\llbracket \boldsymbol{\alpha}_\mathsf{R} + (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_\mathsf{R})(\mathbf{B}\mathbf{r} + \mathbf{a}^\perp \widehat{r}) + \boxed{(\tilde{\omega} \widetilde{u}_{\widetilde{\mathsf{S}}} + \widetilde{v}_\mathsf{R}) \mathbf{a}^\perp \widehat{r}} \right\rrbracket_1 & \text{in } \mathsf{Game}_{2,i,1} \\
\left\llbracket \boldsymbol{\alpha}_\mathsf{R} + \mathbf{a}^\perp \delta_\mathsf{R} + \mathbf{a}^\perp \widehat{v}_i + (\tilde{\omega} \cdot \mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_\mathsf{R})(\mathbf{B}\mathbf{r} + \mathbf{a}^\perp \widehat{r}) + \boxed{(\tilde{\omega} \widetilde{u}_{\widetilde{\mathsf{S}}} + \widetilde{v}_\mathsf{R}) \mathbf{a}^\perp \widehat{r}} \right\rrbracket_1 & \text{in } \mathsf{Game}_{2,i,2}
\end{cases}$$

We focus on the boxed-parts of $(\mathsf{ct}_{1,j}^{[0]})_{j \in [\ell]}$ and tr_1 above. We argue that $\left\{\sigma_j = (\omega^* \widetilde{u}_{\mathsf{S}^*} + \widetilde{v}_{\mathsf{R}^*_{\tau(j)}}) \mathbf{b}^\perp \widehat{s}\right\}_{j \in [\ell]}$ and $\sigma_{\ell+1} = (\tilde{\omega} \widetilde{u}_{\widetilde{\mathsf{S}}} + \widetilde{v}_\mathsf{R}) \mathbf{a}^\perp \widehat{r}$ are uniformly random and independently distributed. This will naturally ensure that $(\mathbf{a}^\perp \delta_\mathsf{R} + \mathbf{a}^\perp \widehat{v}_i)$ is hidden in tr_1 of $\mathsf{Game}_{2,i,2}$ ensuring type-1 and type-2 semi-functional trapdoors are indistinguishable even given the challenge semi-functional ciphertext. We show this independence by explaining $(\sigma_i)_{i \in [\ell+1]}$ as a system of linear equations for following mutually exclusive and exhaustive cases:

Case-1 Consider the case $\mathsf{S}^* \neq \widetilde{\mathsf{S}}$. (Wlog we also consider $\omega^* = \tilde{\omega}$ and $\widetilde{v}_{\mathsf{R}} = \widetilde{v}_{\mathsf{R}^*_{\tau(j)}}$ for some $j \in [\ell]$.)

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{\ell+1} \end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix} \omega^* & 0 & 1 & 0 & \cdots & 0 & \cdots & 0 \\ \omega^* & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \omega^* & 0 & 0 & 0 & \cdots & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \omega^* & 0 & 0 & 0 & \cdots & 0 & \cdots & 1 \\ 0 & \tilde{\omega} & 0 & 0 & \cdots & \underset{\overset{\uparrow}{\tau(j)^{\mathrm{th}}}}{1} & \cdots & 0 \end{bmatrix}_{(\ell+1)\times(\ell+2)} \times \begin{bmatrix} \widetilde{u}_{\mathsf{S}^*} \\ \widetilde{u}_{\widetilde{\mathsf{S}}} \\ \widetilde{v}_{\mathsf{R}^*_{\tau(1)}} \\ \widetilde{v}_{\mathsf{R}^*_{\tau(2)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^*_{\tau(j)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^*_{\tau(\ell)}} \end{bmatrix}_{(\ell+2)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell + 1)$. Since, $\widetilde{u}_{\mathsf{S}^*}, \widetilde{u}_{\widetilde{\mathsf{S}}}, \widetilde{v}_{\mathsf{R}^*_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}^*_{\tau(\ell)}}$ are sampled uniformly at random, $\sigma_1, \ldots, \sigma_{\ell+1}$ are too independent and uniformly random.

Case-2 Consider the case $\mathsf{S}^* = \widetilde{\mathsf{S}} \wedge \omega^* \neq \tilde{\omega}$. Wlog we consider $\widetilde{v}_{\mathsf{R}} = \widetilde{v}_{\mathsf{R}^*_{\tau(j)}}$ for some $j \in [\ell]$.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{\ell+1} \end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix} \omega^* & 1 & 0 & \cdots & 0 & \cdots & 0 \\ \omega^* & 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \omega^* & 0 & 0 & \cdots & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \omega^* & 0 & 0 & \cdots & 0 & \cdots & 1 \\ \tilde{\omega} & 0 & 0 & \cdots & \underset{\overset{\uparrow}{\tau(j)^{\mathrm{th}}}}{1} & \cdots & 0 \end{bmatrix}_{(\ell+1)\times(\ell+1)} \times \begin{bmatrix} \widetilde{u}_{\widetilde{\mathsf{S}}} \\ \widetilde{v}_{\mathsf{R}^*_{\tau(1)}} \\ \widetilde{v}_{\mathsf{R}^*_{\tau(2)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^*_{\tau(j)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^*_{\tau(\ell)}} \end{bmatrix}_{(\ell+1)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell + 1)$. Since, $\widetilde{u}_{\widetilde{\mathsf{S}}}, \widetilde{v}_{\mathsf{R}^*_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}^*_{\tau(\ell)}}$ are sampled uniformly at random and $\omega^* \neq \tilde{\omega}$, $\sigma_1, \ldots, \sigma_{\ell+1}$ are also independent and uniformly random.

Case-3 Consider the case $\mathsf{S}^* = \widetilde{\mathsf{S}} \wedge \omega^* = \tilde{\omega} \wedge \mathsf{R} \notin \mathcal{R}^*$. So, we consider $\widetilde{v}_{\mathsf{R}} \neq \widetilde{v}_{\mathsf{R}^*_{\tau(j)}}$ for all $j \in [\ell]$.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{\ell+1} \end{bmatrix}_{(\ell+1)\times 1} = \begin{bmatrix} \omega^* & 1 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ \omega^* & 0 & 1 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ \omega^* & 0 & 0 & \cdots & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ \omega^* & 0 & 0 & \cdots & 0 & \cdots & 1 & 0 \\ \omega^* & 0 & 0 & \cdots & \underset{\overset{\uparrow}{\tau(j)^{\mathrm{th}}}}{0} & \cdots & 0 & 1 \end{bmatrix}_{(\ell+1)\times(\ell+2)} \times \begin{bmatrix} \widetilde{u}_{\widetilde{\mathsf{S}}} \\ \widetilde{v}_{\mathsf{R}^*_{\tau(1)}} \\ \widetilde{v}_{\mathsf{R}^*_{\tau(2)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^*_{\tau(j)}} \\ \vdots \\ \widetilde{v}_{\mathsf{R}^*_{\tau(\ell)}} \\ \widetilde{v}_{\mathsf{R}} \end{bmatrix}_{(\ell+2)\times 1}$$

The above relation between $\sigma_1, \ldots, \sigma_{\ell+1}$ is represented as a system of the linear equation where the linear transformation has rank $(\ell+1)$. Since, $\widetilde{u}_{\widetilde{\mathsf{S}}}, \widetilde{v}_{\mathsf{R}^*_{\tau(1)}}, \ldots, \widetilde{v}_{\mathsf{R}^*_{\tau(\ell)}}, \widetilde{v}_{\mathsf{R}}$ are sampled uniformly at random, $\sigma_1, \ldots, \sigma_{\ell+1}$ are also independent and uniformly random.

This naturally ensures that $\mathbf{a}^{\perp}\widehat{v}_i$ are hidden in tr_1 of $\mathsf{Game}_{2,i,2}$. Thus, $\mathsf{Game}_{2,i,1}$ and $\mathsf{Game}_{2,i,2}$ are identically distributed. □

Proof of Lemma 10. Same as Proof of Lemma 8. □

Proof of Lemma 11. We make a conceptual change here.

- **Setup**: We sample $\mathfrak{b} \hookleftarrow \{0,1\}$, $\mathbf{A}, \mathbf{B} \hookleftarrow \mathcal{D}_k$ and $\mathbf{a}^{\perp}, \mathbf{b}^{\perp} \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^{\perp}}^{\top}\mathbf{A} = {\mathbf{b}^{\perp}}^{\top}\mathbf{B} = \mathbf{0}$. We publish $\mathsf{pp} = (\llbracket \mathbf{A} \rrbracket_1, \llbracket \mathbf{A} \rrbracket_2, \llbracket \mathbf{B} \rrbracket_1, \llbracket \mathbf{B} \rrbracket_2)$.
- **Query Phase-**I:
  - $O_{\mathsf{pk}}(j)$: We sample $\boldsymbol{\alpha}_j \hookleftarrow \mathbb{Z}_p^{(k+1)}$, $\delta_j \hookleftarrow \mathbb{Z}_p$, $\mathbf{U}_j, \mathbf{V}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$ to set $\mathsf{sk}_j = (\llbracket \boldsymbol{\alpha}_j + \mathbf{a}^{\perp}\delta_j \rrbracket_2, \left\llbracket \mathbf{U}_j^{\top}\mathbf{A} \right\rrbracket_1, \llbracket \mathbf{V}_j\mathbf{B} \rrbracket_2)$ and publish $\mathsf{pk}_j = (\left\llbracket \boldsymbol{\alpha}_j^{\top}\mathbf{A} \right\rrbracket_{\mathrm{T}}, \llbracket \mathbf{U}_j\mathbf{B} \rrbracket_2, \left\llbracket \mathbf{V}_j^{\top}\mathbf{A} \right\rrbracket_1)$.
  - $O_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$: Let $\mathsf{pk}_{\mathcal{R}} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$. To return $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$, we sample a random permutation $\tau : [\ell] \rightarrow [\ell]$, $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$.
  $$\mathsf{ct}_0 = \llbracket \mathbf{A}\mathbf{s} \rrbracket_1, \mathsf{ct}_{1,j} = \left\llbracket (\omega\mathbf{U}_{\mathsf{S}}^{\top} + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^{\top})\mathbf{A}\mathbf{s} \right\rrbracket_1, \kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^{\top}\mathbf{A}\mathbf{s} \right\rrbracket_{\mathrm{T}}.$$
  - $O_{\mathsf{Tr}}(\mathsf{pk}_{\widetilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{pk}_{\mathsf{R}})$: On i^{th} query, we sample $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$ and $\delta_{\mathsf{R}}, \widehat{v}_i \hookleftarrow \mathbb{Z}_p$. We output the trapdoor $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$, where
  $$\mathsf{tr}_0 = \llbracket \mathbf{B}\mathbf{r} \rrbracket_2, \mathsf{tr}_1 = \llbracket \boldsymbol{\alpha}_{\mathsf{R}} + \mathbf{a}^{\perp}\delta_{\mathsf{R}} + \mathbf{a}^{\perp}\widehat{v}_i + (\tilde{\omega}\cdot\mathbf{U}_{\widetilde{\mathsf{S}}} + \mathbf{V}_{\mathsf{R}})\mathbf{B}\mathbf{r} \rrbracket_2.$$
- **Challenge**: Given the challenge $(\mathsf{pk}_{\mathsf{S}^*}, \omega^*, \mathsf{pk}_{\mathcal{R}^*})$ for $\mathsf{pk}_{\mathcal{R}^*} = \{\mathsf{pk}_{\mathsf{R}_1^*}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell^*}\}$, we sample a random permutation $\tau : [\ell] \rightarrow [\ell]$, $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$ and $\widehat{s} \in \mathbb{Z}_p$. Then we compute $\mathsf{Ct}_0^* = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}^{[0]}, \kappa_j^{[0]})_{j\in[\ell]})$ where,
  $$\mathsf{ct}_0 = \left\llbracket \mathbf{A}\mathbf{s} + \mathbf{b}^{\perp}\widehat{s} \right\rrbracket_1$$
  $$\mathsf{ct}_{1,j}^{[0]} = \left\llbracket (\omega^*\mathbf{U}_{\mathsf{S}^*}^{\top} + \mathbf{V}_{\mathsf{R}^*_{\tau(j)}}^{\top})(\mathbf{A}\mathbf{s} + \mathbf{b}^{\perp}\widehat{s}) \right\rrbracket_1$$
  $$\kappa_j^{[0]} = \left\llbracket (\boldsymbol{\alpha}_{\mathsf{R}^*_{\tau(j)}}^{\top} + {\mathbf{a}^{\perp}}^{\top}\delta_{\mathsf{R}^*_{\tau(j)}})(\mathbf{A}\mathbf{s} + \mathbf{b}^{\perp}\widehat{s}) \right\rrbracket_{\mathrm{T}} = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}^*_{\tau(j)}}^{\top}(\mathbf{A}\mathbf{s} + \mathbf{b}^{\perp}\widehat{s}) + \delta_{\mathsf{R}^*_{\tau(j)}}{\mathbf{a}^{\perp}}^{\top}\mathbf{b}^{\perp}\widehat{s} \right\rrbracket_{\mathrm{T}}.$$
  We sample $\mathsf{Ct}_1^* \hookleftarrow \mathcal{CT}$ and return $\mathsf{Ct}_{\mathfrak{b}}^*$ to $\mathcal{A}$.
- **Query Phase-**II: Same as **Query Phase-**I.

Observe that for all $j \in [\ell]$, $\kappa_j^{[0]} = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}^*_{\tau(j)}}^{\top}(\mathbf{A}\mathbf{s} + \mathbf{b}^{\perp}\widehat{s}) + \delta_{\mathsf{R}^*_{\tau(j)}}{\mathbf{a}^{\perp}}^{\top}\mathbf{b}^{\perp}\widehat{s} \right\rrbracket_{\mathrm{T}}$. Since ${\mathbf{a}^{\perp}}^{\top}\mathbf{b}^{\perp} \neq 0$, $\widehat{s} \neq 0$, $\delta_{\mathsf{R}_1^*}, \ldots, \delta_{\mathsf{R}_\ell^*}$ are chosen uniformly at random, $\delta_{\mathsf{R}^*_{\tau(j)}}{\mathbf{a}^{\perp}}^{\top}\mathbf{b}^{\perp}\widehat{s}$ is a uniformly random element for all $j \in [\ell]$. Thus, $\kappa_j^{[0]}$ for all $j \in [\ell]$ of Ct_0^* are uniformly random. □

Proof of Lemma 12. We make a conceptual change here.

- **Setup**: We sample $\mathfrak{b} \hookleftarrow \{0,1\}$, $\mathbf{A}, \mathbf{B} \hookleftarrow \mathcal{D}_k$ and $\mathbf{a}^{\perp}, \mathbf{b}^{\perp} \hookleftarrow \mathbb{Z}_p^{k+1}$ such that ${\mathbf{a}^{\perp}}^{\top}\mathbf{A} = {\mathbf{b}^{\perp}}^{\top}\mathbf{B} = \mathbf{0}$. We publish $\mathsf{pp} = (\llbracket \mathbf{A} \rrbracket_1, \llbracket \mathbf{A} \rrbracket_2, \llbracket \mathbf{B} \rrbracket_1, \llbracket \mathbf{B} \rrbracket_2)$.
- **Query Phase-**I:
  - $O_{\mathsf{pk}}(j)$: We sample $\boldsymbol{\alpha}_j \hookleftarrow \mathbb{Z}_p^{(k+1)}$, $\mathbf{U}_j, \mathbf{V}_j \hookleftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$ to set $\mathsf{sk}_j = (\llbracket \boldsymbol{\alpha}_j \rrbracket_2, \left\llbracket \mathbf{U}_j^{\top}\mathbf{A} \right\rrbracket_1, \llbracket \mathbf{V}_j\mathbf{B} \rrbracket_2)$ and publish $\mathsf{pk}_j = (\left\llbracket \boldsymbol{\alpha}_j^{\top}\mathbf{A} \right\rrbracket_{\mathrm{T}}, \llbracket \mathbf{U}_j\mathbf{B} \rrbracket_2, \left\llbracket \mathbf{V}_j^{\top}\mathbf{A} \right\rrbracket_1)$.
  - $O_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{S}}, \omega, \mathsf{pk}_{\mathcal{R}})$: Let $\mathsf{pk}_{\mathcal{R}} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$. To return $\mathsf{Ct} = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}, \kappa_j)_{j\in[\ell]})$, we sample a random permutation $\tau : [\ell] \to [\ell]$, $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$ where,
$$\mathsf{ct}_0 = \llbracket \mathbf{A}\mathbf{s} \rrbracket_1, \mathsf{ct}_{1,j} = \left\llbracket (\omega\mathbf{U}_{\mathsf{S}}^{\top} + \mathbf{V}_{\mathsf{R}_{\tau(j)}}^{\top})\mathbf{A}\mathbf{s} \right\rrbracket_1, \kappa_j = \left\llbracket \boldsymbol{\alpha}_{\mathsf{R}_{\tau(j)}}^{\top}\mathbf{A}\mathbf{s} \right\rrbracket_{\mathrm{T}}.$$
  - $O_{\mathsf{Tr}}(\mathsf{pk}_{\tilde{\mathsf{S}}}, \tilde{\omega}, \mathsf{pk}_{\mathsf{R}})$: On i^{th} query, we sample $\mathbf{r} \hookleftarrow \mathbb{Z}_p^k$ and $\mathbf{v}_i \hookleftarrow \mathbb{Z}_p^{k+1}$. We output the trapdoor $\mathsf{Tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$, where
$$\mathsf{tr}_0 = \llbracket \mathbf{B}\mathbf{r} \rrbracket_2, \mathsf{tr}_1 = \llbracket \boldsymbol{\alpha}_{\mathsf{R}} + \tilde{\omega}\cdot\mathbf{U}_{\tilde{\mathsf{S}}}\mathbf{B}\mathbf{r} + \mathbf{v}_i \rrbracket_2.$$
- **Challenge**: Given the challenge $(\mathsf{pk}_{\mathsf{S}^*}, \omega^*, \mathsf{pk}_{\mathcal{R}^*})$ for $\mathsf{pk}_{\mathcal{R}^*} = \{\mathsf{pk}_{\mathsf{R}_1^*}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell^*}\}$, we sample a random permutation $\tau : [\ell] \to [\ell]$, $\mathbf{s} \hookleftarrow \mathbb{Z}_p^k$ and $\widehat{s} \in \mathbb{Z}_p$. Then we compute $\mathsf{Ct}_0^* = (\mathsf{ct}_0, (\mathsf{ct}_{1,j}^{[0]}, \kappa_j^{[0]})_{j\in[\ell]})$ where,
$$\mathsf{ct}_0 = \left\llbracket \mathbf{A}\mathbf{s} + \mathbf{b}^{\perp}\widehat{s} \right\rrbracket_1, \mathsf{ct}_{1,j}^{[0]} = \left\llbracket \omega^*\mathbf{U}_{\mathsf{S}^*}^{\top}(\mathbf{A}\mathbf{s} + \mathbf{b}^{\perp}\widehat{s}) + \mathbf{z}_j \right\rrbracket_1, \kappa_j^{[0]} \hookleftarrow \mathbb{G}_{\mathrm{T}}.$$
  We sample $\mathsf{Ct}_1^* \hookleftarrow \mathcal{CT}$ and return $\mathsf{Ct}_{\mathfrak{b}}^*$ to $\mathcal{A}$.
- **Query Phase-**II: Same as **Query Phase-**I.

We first argue that the modified trapdoors are distributed the same way as they were in Game_3. For that, we argue that $\mathbf{V}_{\mathsf{R}}\mathbf{B}\mathbf{r} + \mathbf{a}^{\perp}\widehat{v}_i$ is uniformly random. To see this, we left multiply this with $\begin{pmatrix} \mathbf{A}^{\top} \\ {\mathbf{b}^{\perp}}^{\top} \end{pmatrix}$. Then,
$$\begin{pmatrix} \mathbf{A}^{\top} \\ {\mathbf{b}^{\perp}}^{\top} \end{pmatrix} \cdot (\mathbf{V}_{\mathsf{R}}\mathbf{B}\mathbf{r} + \mathbf{a}^{\perp}\widehat{v}_i) = \begin{pmatrix} \mathbf{A}^{\top}\mathbf{V}_{\mathsf{R}}\mathbf{B}\mathbf{r} \\ {\mathbf{b}^{\perp}}^{\top}\mathbf{V}_{\mathsf{R}}\mathbf{B}\mathbf{r} + {\mathbf{b}^{\perp}}^{\top}\mathbf{a}^{\perp}\widehat{v}_i \end{pmatrix}$$
such that ${\mathbf{b}^{\perp}}^{\top}\mathbf{a}^{\perp}\widehat{v}_i$ is an uniformly random value. As $\begin{pmatrix} \mathbf{A}^{\top} \\ {\mathbf{b}^{\perp}}^{\top} \end{pmatrix}$ spans the space, $\mathbf{V}_{\mathsf{R}}\mathbf{B}\mathbf{r} + \mathbf{a}^{\perp}\widehat{v}_i$ is uniformly random. Therefore replacing this with uniformly random $\mathbf{v}_i$ is invisible to $\mathcal{A}$.

Then we argue that, for all $j \in [\ell]$, $\mathbf{V}_{\mathsf{R}_{\tau(j)}^*}(\mathbf{A}\mathbf{s} + \mathbf{b}^{\perp}\widehat{s})$ in $\mathsf{ct}_{1,j}^{[0]}$ of Game_3 are uniformly random. Now, two mutual exclusive situations can happen that are exhaustive for all $j \in [\ell]$,

1. No trapdoor query involved $\mathsf{R}^*_{\tau(j)}$: $\mathbf{V}_{\mathsf{R}^*_{\tau(j)}}$ are completely hidden from $\mathcal{A}$.
2. If $\mathsf{R}^*_{\tau(j)}$ was used for trapdoor queries: $\mathbf{V}_{\mathsf{R}^*_{\tau(j)}}\mathbf{B}$ are completely hidden in the trapdoors by above argument.

Thus, $\mathcal{A}$ only gets $\mathbf{V}^\top_{\mathsf{R}^*_{\tau(j)}}\mathbf{A}$ from $\mathsf{pk}_{\mathsf{R}^*_{\tau(j)}}$ and any ciphertext query made on $\mathsf{R}^*_{\tau(j)}$. As $\mathbf{b}^\perp$ is outside the span of $\mathbf{A}$, $\mathbf{V}^\top_{\mathsf{R}^*_{\tau(j)}}(\mathbf{As}+\mathbf{b}^\perp\widehat{s})$ is independent from $\mathsf{pk}_{\mathsf{R}^*_{\tau(j)}}$. As $\mathbf{V}^\top_{\mathsf{R}^*_{\tau(j)}}(\mathbf{As}+\mathbf{b}^\perp\widehat{s})$ for all $j \in [\ell]$ are independent from $\mathbf{V}^\top_{\mathsf{R}^*_{\tau(j)}}\mathbf{A}$, we replace them by $\mathbf{z}_j \hookleftarrow \mathbb{Z}_p^{k+1}$. □

Proof of Lemma 13. Finally, we state that $\mathsf{Adv}^{\mathsf{Game}_4}_{\mathcal{A}}(\lambda) = 0$. This is because, all $\{\mathbf{z}_j\}_{j\in[\ell]}$ are uniformly random vectors that hide $\omega^* \cdot \mathbf{U}^\top_{\mathsf{S}^*}(\mathbf{As}+\mathbf{b}^\perp\widehat{s})$ in $\mathsf{ct}^{[0]}_{1,j}$. □

5 Comparison

We compare our construction with the state of the art [20]. The primary difference between the two schemes is use of Random Oracle model. We reiterate that our construction is a standard model construction and does not use assumptions like hash functions are random.

Functionally both the schemes are quite similar except the shortcomings of the security models in [20] that we mention in Sect. 3.3. We also mention here we do not need KGC to stay online throughout and users can generate their own key-pairs unlike [20].

Tables 3 and 4 provide comparisons of computation cost and communication overheads. Here, ℓ denotes the size of the privileged set and k is a constant that determines the level of security *bil*-$\mathcal{D}_k$-matDH assumption provides. In the following two tables, $|\mathbb{Z}_p|$ refers to the element size of field $\mathbb{Z}_p$, $|\mathbb{G}|$ (resp. $|\mathbb{G}_1|, |\mathbb{G}_2|, |\mathbb{G}_\mathrm{T}|$) captures bit-length representation of any element from $\mathbb{G}$ (resp. $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}$). As exponentiation and pairings dominate the execution time, we focus on them. $\mathsf{G_e}$ refers to exponentiation, $\mathsf{G_p}$ refers to pairing.

Table 3. Computation Cost Comparison

	SrchEnc	Trapdoor	Test
[20]	$(2\ell+5)\mathsf{G_e}+\ell\mathsf{G_p}$	$\mathsf{G_e}+\mathsf{G_p}$	$(\ell+4)\mathsf{G_e}+2\mathsf{G_p}$
Our	$(2\ell+1)k\mathsf{G_e}+\ell\mathsf{G_p}$	$2k\mathsf{G_e}$	$(\ell+1)\mathsf{G_p}$

Table 4. Communication Complexity Comparison

	pp	pk_i	sk_i	Ct	Tr
[20]	$4\lvert\mathbb{G}\rvert$	$\lvert\mathbb{G}\rvert$	$\lvert\mathbb{Z}_p\rvert$	$\lvert\mathbb{Z}_p\rvert+(\ell+2)\lvert\mathbb{G}\rvert$	$\lvert\mathbb{Z}_p\rvert$
Our	$2(k+1)k\lvert\mathbb{G}_1\rvert$ $+2(k+1)k\lvert\mathbb{G}_2\rvert$	$(k+1)k\lvert\mathbb{G}_1\rvert$ $+(k+1)k\lvert\mathbb{G}_2\rvert$ $+k\lvert\mathbb{G}_\mathrm{T}\rvert$	$(k+1)k\lvert\mathbb{G}_1\rvert$ $+(k+1)k\lvert\mathbb{G}_2\rvert$ $+(k+1)\lvert\mathbb{G}_\mathrm{T}\rvert$	$(\ell+1)(k+1)\lvert\mathbb{G}_1\rvert$ $+\ell\lvert\mathbb{G}_\mathrm{T}\rvert$	$2(k+1)\lvert\mathbb{G}_2\rvert$

Note that, $|\mathsf{Ct}|$ can be further improved by using a universal hash function that maps $\mathbb{G}_\mathrm{T} \rightarrow \mathbb{Z}_p$. In that case, $|\mathsf{Ct}| = (\ell+1)(k+1)|\mathbb{G}_1| + \ell\mathbb{Z}_p$.

From the above two tables (Tables 3 and 4) we see that both the constructions achieve efficiency similar asymptotically. However, in concrete terms, ours look a bit shabby. That being said, we must reiterate that our construction is instantiated in type-3 settings whereas [20] achieves the construction is less efficient type-1 settings and therefore, a concrete comparison is difficult. We believe the stronger security guarantees of our construction without hampering the efficiency argues its utility quite convincingly.

6 Conclusion

This paper revamps existing BAEKS security models of [20] to consider ciphertext and trapdoor security from the lens of sender, receiver, and keyword privacy. We further propose a consistency definition for BAEKS. Following this, we propose a new *statistically consistent* construction of BAEKS which we prove to be secure in the standard model. More precisely, our novel BAEKS construction achieves adaptive security (in terms of both ciphertext security and trapdoor security) under the standard assumption bilateral Matrix Diffie-Hellman (bil-$\mathcal{D}_k$-matDH). Interestingly, our construction still achieves asymptotic efficiency similar to that of [20]. For future work, one might aim for a new construction that achieves more robust security in the presence of malicious adversaries.

A Security Models of [20]

In [20], Liu et al. introduced broadcast authenticated encryption with keyword search (BAEKS) as an extension of public-key authenticated encryption with keyword search (PAEKS) [18]. To capture the challenges in case of this new primitive, [20] introduced several security games. We reproduce the games from [20] next for completeness. They introduced four security games– two games to capture the security of trapdoors and two to capture the security of ciphertexts.

Trapdoor Privacy. Informally, trapdoor privacy captures trapdoors do not leak the keywords encoded. The formal security game next is reproduced verbatim from [20].

- **Setup**: Given security parameter, the challenger $\mathcal{C}$ sends the public parameter pp, the challenge sender's public key ($\mathsf{pk}_{\mathsf{S}^*}$) and the challenge receiver's public key ($\mathsf{pk}_{\mathsf{R}^*}$) to the adversary $\mathcal{A}$.
- **Query Phase-**I:
  - *Hash Queries:* $\mathcal{C}$ responds to hash queries with random numbers.
  - *Ciphertext Query:* Given a keyword ω, a receiver set's public keys $\mathcal{R} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$, $\mathcal{C}$ computes a ciphertext w.r.t $\mathsf{sk}_{\mathsf{S}^*}$, ω and $\mathcal{R}$ and returns it to $\mathcal{A}$.

- *Trapdoor Query:* Given a keyword $\tilde{\omega}$, a sender's public key $\mathsf{pk}_{\tilde{\mathsf{S}}}$, a chosen public key $\mathsf{pk}_{\mathsf{R}_i} \in \mathcal{R}$, it computes a trapdoor Tr w.r.t. $\mathsf{pk}_{\tilde{\mathsf{S}}}$, $\tilde{\omega}$ and $\mathsf{pk}_{\mathsf{R}_i}$, returns it to $\mathcal{A}$.

- **Challenge**: $\mathcal{A}$ chooses two keywords $(\tilde{\omega}_0^*, \tilde{\omega}_1^*)$ such that $(\tilde{\omega}_0^*, \mathcal{R})$ and $(\tilde{\omega}_1^*, \mathcal{R})$ have not been queried for ciphertexts where $\mathsf{pk}_{\mathsf{R}^*} \in \mathcal{R}$, and $(\tilde{\omega}_0^*, \mathsf{pk}_{\mathsf{S}^*})$ and $(\tilde{\omega}_1^*, \mathsf{pk}_{\mathsf{S}^*})$ have not been queried for trapdoors and sends them to $\mathcal{C}$. $\mathcal{C}$ randomly chooses a bit $b \hookleftarrow \{0,1\}$ and provides $\mathcal{A}$ with a trapdoor $\mathsf{Tr}^* \leftarrow \mathsf{Trapdoor}(\mathsf{pk}_{\mathsf{S}^*}, \tilde{\omega}_b^*, \mathsf{sk}_{\mathsf{R}^*})$ and returns it to $\mathcal{A}$.
- **Query Phase-II**: Similar to **Query Phase-I** maintaining natural restrictions.
- **Guess**: $\mathcal{A}$ guesses a bit b' and wins if $b = b'$.

Ciphertext Indistinguishability. Informally, ciphertext indistinguishability captures ciphertexts do not leak the keywords encoded. The formal security game is as follows.

- **Setup**: Given security parameter, the challenger $\mathcal{C}$ sends the public parameter pp, the challenge sender's public key $(\mathsf{pk}_{\mathsf{S}^*})$ and the challenge receiver's set public key $(\mathcal{R}^* = \{\mathsf{pk}_{\mathsf{R}_1^*}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell^*}\})$ to the adversary $\mathcal{A}$.
- **Query Phase-I**:
  - *Hash Queries:* $\mathcal{C}$ responds to hash queries with random numbers.
  - *Ciphertext Query:* Given a keyword ω, a receiver set's public keys $\mathcal{R} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$, $\mathcal{C}$ computes a ciphertext w.r.t $\mathsf{sk}_{\mathsf{S}^*}$, ω and $\mathcal{R}$ and returns it to $\mathcal{A}$.
  - *Trapdoor Query:* Given a keyword $\tilde{\omega}$, a sender's public key $\mathsf{pk}_{\tilde{\mathsf{S}}}$, a chosen public key $\mathsf{pk}_{\mathsf{R}_i} \in \mathcal{R}$, it computes a trapdoor Tr w.r.t. $\mathsf{pk}_{\tilde{\mathsf{S}}}$, $\tilde{\omega}$ and $\mathsf{sk}_{\mathsf{R}_i}$, returns it to $\mathcal{A}$.
- **Challenge**: $\mathcal{A}$ chooses two keywords (ω_0^*, ω_1^*) such that $(\omega_0^*, \mathsf{pk}_{\mathsf{S}^*})$ and $(\omega_1^*, \mathsf{pk}_{\mathsf{S}^*})$ have not been queried for trapdoors and sends them to $\mathcal{C}$. $\mathcal{C}$ randomly chooses a bit $b \hookleftarrow \{0,1\}$ and provides $\mathcal{A}$ with a ciphertext $\mathsf{Ct}^* \leftarrow \mathsf{SrchEnc}(\mathsf{sk}_{\mathsf{S}^*}, \omega_b^*, \mathcal{R}^*)$ and returns it to $\mathcal{A}$.
- **Query Phase-II**: Similar to **Query Phase-I** maintaining natural restrictions.
- **Guess**: The adversary guesses a bit b' and wins if $b = b'$.

Anonymity. Informally, anonymity captures ciphertexts do not leak the receiver set encoded. The formal security game is as follows.

- **Setup**: Given security parameter, the challenger $\mathcal{C}$ sends the public parameter pp, the challenge sender's public key $(\mathsf{pk}_{\mathsf{S}^*})$ and the challenge receiver's set public key $(\mathcal{R}_0^* = \{\mathsf{pk}_{\mathsf{R}_0^*}, \mathsf{pk}_{\mathsf{R}_2^*}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell^*}\})$, $(\mathcal{R}_1^* = \{\mathsf{pk}_{\mathsf{R}_1^*}, \mathsf{pk}_{\mathsf{R}_2^*}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell^*}\})$ to the adversary $\mathcal{A}$.
- **Query Phase-I**:
  - *Hash Queries:* $\mathcal{C}$ responds to hash queries with random numbers.
  - *Ciphertext Query:* Given a keyword ω, a receiver set's public keys $\mathcal{R} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$, $\mathcal{C}$ computes a ciphertext w.r.t $\mathsf{sk}_{\mathsf{S}^*}$, ω and $\mathcal{R}$ and returns it to $\mathcal{A}$.

  - *Trapdoor Query:* Given a keyword $\tilde{\omega}$, a sender's public key $\mathsf{pk}_{\tilde{\mathsf{S}}}$, a chosen public key from $\{\mathsf{pk}_{\mathsf{R}_0}, \mathsf{pk}_{\mathsf{R}_1}\}$, it computes a trapdoor Tr w.r.t. $\mathsf{pk}_{\tilde{\mathsf{S}}}$, $\tilde{\omega}$ and $\mathsf{pk}_{\mathsf{R}_i}$ for $i \in \{0, 1\}$, returns it to $\mathcal{A}$.
- **Challenge**: $\mathcal{A}$ chooses a keyword ω^* such that $(\omega^*, \mathsf{pk}_{\mathsf{S}^*})$ has not been queried for trapdoors and sends them to $\mathcal{C}$. $\mathcal{C}$ randomly chooses a bit $b \hookleftarrow \{0, 1\}$ and provides $\mathcal{A}$ with $\mathsf{Ct}^* \leftarrow \mathsf{SrchEnc}(\mathsf{sk}_{\mathsf{S}^*}, \omega^*, \mathsf{R}_b^*)$ and returns it to $\mathcal{A}$.
- **Query Phase-II**: Similar to **Query Phase-I** maintaining natural restrictions.
- **Guess**: The adversary guesses a bit b' and wins if $b = b'$.

Trapdoor Anonymity. Informally, trapdoor anonymity captures trapdoors do not leak the receiver information encoded. The formal security game is as follows.

- **Setup**: Given security parameter, the challenger $\mathcal{C}$ sends the public parameter pp, the challenge sender's public key $(\mathsf{pk}_{\mathsf{S}^*})$ and two challenge receiver's public key $(\mathsf{pk}_{\mathsf{R}_0^*}, \mathsf{pk}_{\mathsf{R}_1^*})$ to the adversary $\mathcal{A}$.
- **Query Phase-I**:
  - *Hash Queries:* $\mathcal{C}$ responds to hash queries with random numbers.
  - *Ciphertext Query:* Given a keyword ω, a receiver set's public keys $\mathcal{R} = \{\mathsf{pk}_{\mathsf{R}_1}, \ldots, \mathsf{pk}_{\mathsf{R}_\ell}\}$, $\mathcal{C}$ computes a ciphertext w.r.t $\mathsf{sk}_{\mathsf{S}^*}$, ω and $\mathcal{R}$ and returns it to $\mathcal{A}$.
  - *Trapdoor Query:* Given a keyword $\tilde{\omega}$, a sender's public key $\mathsf{pk}_{\tilde{\mathsf{S}}}$, a chosen public key from $\{\mathsf{pk}_{\mathsf{R}_0}, \mathsf{pk}_{\mathsf{R}_1}\}$, it computes a trapdoor Tr w.r.t. $\mathsf{pk}_{\tilde{\mathsf{S}}}$, $\tilde{\omega}$ and $\mathsf{pk}_{\mathsf{R}_i}$ for $i \in \{0, 1\}$, returns it to $\mathcal{A}$.
- **Challenge**: $\mathcal{A}$ chooses a keyword $\tilde{\omega}^*$ such that $(\tilde{\omega}_0^*, \mathsf{pk}_{\mathsf{S}^*})$ has not been queried for trapdoors and $(\tilde{\omega}^*, \mathcal{R})$ has not been queried for ciphertexts where $\mathsf{R}_0^*, \mathsf{R}_1^*$ have different inclusion relationships with $\mathcal{R}$, and sends them to $\mathcal{C}$. $\mathcal{C}$ randomly chooses a bit $b \hookleftarrow \{0, 1\}$ and provides $\mathcal{A}$ with a trapdoor $\mathsf{Tr}^* \leftarrow \mathsf{Trapdoor}(\mathsf{pk}_{\mathsf{S}^*}, \tilde{\omega}^*, \mathsf{sk}_{\mathsf{R}_b^*})$ and returns it to $\mathcal{A}$.
- **Query Phase-II**: Similar to **Query Phase-I** maintaining natural restrictions.
- **Guess**: $\mathcal{A}$ guesses a bit b' and wins if $b = b'$.

References

1. Abdalla, M., et al.: Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extensions. J. Cryptol. **21**(3), 350–391 (2007). https://doi.org/10.1007/s00145-007-9006-6
2. Attrapadung, N., Furukawa, J., Imai, H.: Forward-secure and searchable broadcast encryption with short ciphertexts and private keys. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 161–177. Springer, Heidelberg (2006). https://doi.org/10.1007/11935230_11
3. Baek, J., Safavi-Naini, R., Susilo, W.: Public key encryption with keyword search revisited. In: Gervasi, O., Murgante, B., Laganà, A., Taniar, D., Mun, Y., Gavrilova, M.L. (eds.) ICCSA 2008. LNCS, vol. 5072, pp. 1249–1259. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69839-5_96

4. Boneh, D., Boyen, X.: Efficient selective-id secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_14
5. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_26
6. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_30
7. Boneh, D., Gentry, C., Waters, B.: Collusion resistant broadcast encryption with short ciphertexts and private keys. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 258–275. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_16
8. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_29
9. Byun, J.W., Rhee, H.S., Park, H.-A., Lee, D.H.: Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In: Jonker, W., Petković, M. (eds.) SDM 2006. LNCS, vol. 4165, pp. 75–83. Springer, Heidelberg (2006). https://doi.org/10.1007/11844662_6
10. Chatterjee, S., Mukherjee, S.: Keyword search meets membership testing: adaptive security from SXDH. In: Chakraborty, D., Iwata, T. (eds.) INDOCRYPT 2018. LNCS, vol. 11356, pp. 21–43. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05378-9_2
11. Chen, J., Gay, R., Wee, H.: Improved dual system ABE in prime-order groups via predicate encodings. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 595–624. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_20
12. Chi, T., Qin, B., Zheng, D.: An efficient searchable public-key authenticated encryption for cloud-assisted medical internet of things. Wirel. Commun. Mobile Comput. **2020**, 8816172 (2020). https://doi.org/10.1155/2020/8816172
13. Emura, K.: Generic construction of public-key authenticated encryption with keyword search revisited: stronger security and efficient construction. In: ASIA Public-Key Cryptography Workshop. pp. 39–49. APKC '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3494105.3526237
14. Emura, K., Miyaji, A., Rahman, M.S., Omote, K.: Generic constructions of secure-channel free searchable encryption with adaptive security. Secur. Commun. Netw. **8**(8), 1547–1560 (2015). https://doi.org/10.1002/sec.1103
15. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for diffie–hellman assumptions. J. Cryptol. **30**(1), 242–288 (2015). https://doi.org/10.1007/s00145-015-9220-6
16. Fang, L., Susilo, W., Ge, C., Wang, J.: A secure channel free public key encryption with keyword search scheme without random oracle. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 248–258. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10433-6_16
17. Gentry, C., Waters, B.: Adaptive security in broadcast encryption systems (with short ciphertexts). In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 171–188. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_10

18. Huang, Q., Li, H.: An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. Inf. Sci. 403–404, 1–14 (2017). https://doi.org/10.1016/j.ins.2017.03.038, https://www.sciencedirect.com/science/article/pii/S0020025516321090
19. Jutla, C.S., Roy, A.: Shorter quasi-adaptive NIZK proofs for linear subspaces. J. Cryptol. **30**(4), 1116–1156 (2016). https://doi.org/10.1007/s00145-016-9243-7
20. Liu, X., He, K., Yang, G., Susilo, W., Tonien, J., Huang, Q.: Broadcast authenticated encryption with keyword search. In: Baek, J., Ruj, S. (eds.) ACISP 2021. LNCS, vol. 13083, pp. 193–213. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90567-5_10
21. Pan, X., Li, F.: Public-key authenticated encryption with keyword search achieving both multi-ciphertext and multi-trapdoor indistinguishability. J. Syst. Architect. **115**, 102075 (2021). https://doi.org/10.1016/j.sysarc.2021.102075, https://www.sciencedirect.com/science/article/pii/S1383762121000643
22. Waters, B.: Dual system encryption: realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_36
23. Yau, W.-C., Heng, S.-H., Goi, B.-M.: Off-line keyword guessing attacks on recent public key encryption with keyword search schemes. In: Rong, C., Jaatun, M.G., Sandnes, F.E., Yang, L.T., Ma, J. (eds.) ATC 2008. LNCS, vol. 5060, pp. 100–105. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69295-9_10

Modular Design of KEM-Based Authenticated Key Exchange

Colin Boyd, Bor de Kock(✉), and Lise Millerjord

NTNU – Norwegian University of Science and Technology, Trondheim, Norway
{colin.boyd,bor.dekock,lise.millerjord}@ntnu.no

Abstract. A key encapsulation mechanism (KEM) is a basic building block for key exchange which must be combined with long-term keys in order to achieve authenticated key exchange (AKE). Although several KEM-based AKE protocols have been proposed, KEM-based modular building blocks are not available. We provide a KEM-based authenticator and a KEM-based protocol in the Authenticated Links model (AM), in the terminology of Canetti and Krawczyk (2001). Using these building blocks we achieve a set of generic AKE protocols. By instantiating these with post quantum primitives we are able to propose several new post-quantum secure AKE protocols.

1 Introduction

Authenticated key exchange (AKE) is a fundamental tool for establishing secure communications. An important component in the design of AKE protocols is Diffie–Hellman (DH) key exchange, due to its versatility and potential for providing security properties such as forward secrecy. Today many real-world AKE protocols are based on DH implementations, typically in elliptic curve groups; examples include TLS, IPSec, WireGuard and the generic Noise Framework.

The looming threat of quantum computers has brought about an increasingly pressing need to find post-quantum secure replacements for DH, which itself is well known to be broken by Shor's quantum algorithm for finding discrete logarithms [5]. In the absence of many promising candidates for a post-quantum secure direct DH replacement, designs for post-quantum AKE have tended to make use of key encapsulation mechanisms (KEM). This approach aligns well with the research literature where many post-quantum candidate KEMs have been proposed and also with the prominent NIST post-quantum cryptography competition [1] which requests primitives of only two types, namely a KEM or a digital signature. Although DH can be framed as a KEM, DH has special properties which prevent KEMs from being used as a drop-in replacement for DH. For example, DH has the property that two parties can generate their DH

Boyd and Millerjord are supported by the Research Council of Norway under Project No. 288545. Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. A full version of this work is available on the IACR ePrint Archive under number 2023/167 [8].

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 553–579, 2023.
https://doi.org/10.1007/978-3-031-35486-1_24

shares completely independently; this cannot be achieved in general with KEMs, but rather one party must wait for the other party's input.

Achieving the *authenticated* part of AKE has traditionally been done by applying a digital signature scheme on the messages of a key exchange protocol, but authentication can also be achieved in different ways, which can be advantageous for several reasons: for instance to achieve a speed-up or use less memory, as other works have demonstrated [23].

Although in 2022 the NIST competition for post-quantum secure cryptography (PQ or PQ-crypto) has led to the standardization of several signature schemes, only one KEM was standardized along with an open call for more schemes to be proposed [1]. To achieve authentication without depending on this one scheme is a desirable property. One of the main motivations for our work is to be flexible in the use of cryptographic primitives so that as the security of post-quantum KEMs or signatures becomes better understood, and as new primitives are designed, it is easy to swap in and out different ones.

1.1 Modular Design of AKE

Over the past decade, several key exchange protocols using post-quantum candidate KEMs have been proposed, both authenticated [13] and unauthenticated [7,14,22]. Some of these protocols have been proposed based on specific KEM constructions and the security proofs (where available) relate to specific computational assumptions. These are essential constructions for instantiating protocols using abstract primitives, but when using specific constructions as the basis of security for AKE there is a loss of cryptographic agility. Our goal in this work is to design generic AKE protocols where we can be as flexible as possible with regard to choice of specific KEM instantiations and how they are used.

Our protocol designs are based on the modular approach of Bellare, Canetti and Krawczyk [3,17] (hereafter referred to as BCK98) and Canetti and Krawczyk [9] (hereafter referred to as CK01). This approach entails defining protocols which are secure in a world which is ideally authenticated and then compiling these protocols with *authenticators* to achieve protocols secure in a world where adversaries completely control the network. A brief introduction to this modular approach is given in Sect. 2.2.

A significant benefit of the modular approach is the ability to "mix-and-match" different components and to use different concrete instances of the same component within one protocol instantiation. This leads to a plethora of different concrete protocols with varying performance characteristics.

We remark that there already exist several protocol designs which are generic in the sense that they can use any specific (secure) instance of different cryptographic primitives such as non-interactive key exchange (NIKE), signatures and/or KEMs [4,15,19]. However, such designs do not allow generic mixing of different generic primitives as can be done with the modular approach. For example, the modular approach can be used to replace a digital signature authentication method which a MAC-based method in the case that a pre-shared key is available in a particular application.

1.2 Contributions

We regard the following as the main contributions of this paper:

1. We develop a new KEM-based authenticator and prove its security as a valid authenticator in the CK01 definition, relying on the established CCA-security definition for KEMs.
2. We frame the well-known method of using an ephemeral KEM as a DH replacement as a protocol in the authenticated-links model (AM) of CK01 and prove its security in that model, relying on the established CPA-security definition for KEMs.
3. We derive efficient and secure generic AKE protocols which can be instantiated with any appropriately secure KEMs and also matched with other primitives such as signatures. Some of these generic protocols are completely new, allowing new instantiations of concrete protocols.

1.3 Related Work

In 2017, De Saint Guilhem, Smart and Warinschi [12] presented a generic transformation to convert any two-round forward-secret, but only passively secure, key agreement protocol into a three-round authenticated key agreement protocol. Recognising the value of avoiding signatures for authentication in the post-quantum setting, their transformation makes use of generic CCA-secure public key encryption and a secure MAC. While the approach of De Saint Guilhem et al. has clear parallels with ours, they rely on encryption rather than the often more efficient notion of KEMs. Moreover, they do not allow mixing of different authentication methods as we do, nor provide KEM-based concrete passively secure protocols. Furthermore, their proofs require a key derivation function modelled as a random oracle. Interestingly, they dismiss the CK01 modular approach stating that it necessarily results in increased number of rounds; below we will explain why this is not the case.

Several recent works show that KEM-based approaches are suitable for replacing signatures in real-world applications. The KEMTLS protocol of Schwabe, Stebila and Wiggers [23] is for instance a complete reworking of the TLS 1.3 handshake without using signatures, showing that this would in theory require only half the bandwidth compared to a classical approach—with additional improvements to be gained if the public keys are exchanged in advance [24]. Some of these theoretical improvements turned out to be less impactful when looking at a real-world implementation [10].

Using KEM as a building block for AKE is also done in some other purpose-specific works: examples of this include Post-Quantum Noise [2], FSXY [15] and Post-Quantum WireGuard [18]. These are generic in the sense that any suitable KEMs can be used, but they do not allow the flexibility of different authenticators that we obtain. Specifically, they do not provide re-usable and interchangable components for passive security and for authentication.

There exist many formal security models for AKE, amongst which several are incomparable [11] in the sense that any one model is often neither stronger nor weaker than another. The modular approach that we use [9] achieves security in the well-established model known widely as the CK-model. This encompasses fundamental security properties of session key indistinguishability against active attackers who can obtain non-target session keys and adaptively corrupt non-target parties. Forward secrecy is also captured. This model can be adapted [21] if other security properties, such as ephemeral key leakage, are desirable.

2 Background

The main goal of this section is to present the background necessary to understand the modular approach of (Bellare), Canetti and Krawczyk [3,9]. This includes our method to optimise, in a rigorous way, protocols obtained through the approach.

2.1 Standard Definitions

We make use of standard definitions such as KEM, MAC, signatures etc. These can be found in, for example, the textbook of Katz and Lindell [20].

Definition 1. *A key encapsulation mechanism* (KEM) *is a tuple of PPT algorithms* (Gen, Encap, Decap) *such that:*

1. *The key generation algorithm* Gen *takes the security parameter* 1^n *and outputs a public-/private-key pair (sk, pk):* $(sk, pk) \leftarrow \mathsf{Gen}(1^n)$.
2. *The encapsulation algorithm* Encap *takes as input a public key* pk. *It outputs a ciphertext* c *and a key* $k \in \{0,1\}^{l(n)}$: $(c, k) \leftarrow \mathsf{Encap}(pk)$.
3. *The deterministic decapsulation algorithm* Decap *takes as input a private key* sk *and a ciphertext* c, *and outputs a key* k *or the special symbol* $\perp$ *denoting failure:* $k \leftarrow \mathsf{Decap}(sk, c)$.

We require correctness *from the* KEM: *If* $(sk, pk) \leftarrow \mathsf{Gen}(1^n)$ *and* $(c, k) \leftarrow \mathsf{Encap}_{pk}(1^n)$, *and* $k' \leftarrow \mathsf{Decap}_{sk}(c)$, *then* $k' = k$ *except with negligible probability.*

Furthermore we show the CPA (resp. CCA) indistinguishability experiment(s) for KEMs.

Definition 2. *The* CPA $\boxed{\textit{resp.}\ \mathsf{CCA}}$ *indistinguishability experiment proceeds as follows:*

1. *The key generation algorithm is run:* $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$.
2. *The encapsulation algorithm is run:* $(c, k) \leftarrow \mathsf{Encap}(pk)$, *with* $k \in \{0,1\}^n$.
3. *A uniform bit* $b \in \{0,1\}$ *is chosen. If* $b = 0$, *set* $k' = k$. *Otherwise, if* $b = 1$, *choose a uniform* $k' \in \{0,1\}^n$.
4. *The experiments outputs* (pk, c, k') *to* $\mathcal{A}$.

$\mathcal{A}$ is also given access to a decapsulation oracle, $\mathsf{Decap}_{sk}(\cdot)$, *but cannot query the decapsulation oracle on the ciphertext c.*

5. *$\mathcal{A}$ outputs a bit b'. If $b' = b$, $\mathcal{A}$ wins and the experiment outputs 1. Otherwise, $\mathcal{A}$ loses and the experiment outputs 0.*

The advantage of the adversary $\mathcal{A}$ in the CPA $\boxed{\mathsf{CCA}}$ *experiment is defined to be:*

$$\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{CPA}\boxed{\mathsf{CCA}}}(\mathcal{A}) = 2 \cdot \left|\Pr\left[b' = b\right] - 1/2\right|.$$

Definition 3. *A* message authentication code *or* MAC *consists of three probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{MacVer})$ *such that:*

1. *The* key-generation algorithm Gen *takes as input the security parameter* 1^n *and outputs a key k with $|k| \geq n$.*
2. *The* tag-generation algorithm Mac *takes as input a key k and a message $m \in \{0,1\}^*$, and outputs a tag t. Since this algorithm may be randomized, we write this as $t \leftarrow$* $\mathsf{Mac}_k(m)$.
3. *The deterministic* verification algorithm MacVer *takes as input a key k, a message m and a tag t. It outputs a bit b, with $b = 1$ meaning* valid *and $b = 0$ meaning* invalid. *We write this as* $b := \mathsf{MacVer}_k(m, t)$.

It is required that for every n, every key k and output by $\mathsf{Gen}(1^n)$ *and every $m \in \{0,1\}$, it holds that* $\mathsf{MacVer}_k(m, \mathsf{Mac}(m)) = 1$.

Definition 4. *The existential unforgeability under chosen message attacks* ($\mathsf{EUF\text{-}CMA}$) *experiment for* $\mathsf{MAC}(\mathsf{Gen}, \mathsf{Mac}, \mathsf{MacVer})$ *proceeds as follows:*

1. *A key is generated: $k \leftarrow$* $\mathsf{Gen}(1^n)$.
2. *The adversary $\mathcal{A}$ gets oracle access to* $\mathsf{Mac}_k(\cdot)$. *Let Q be the set of all queries $\mathcal{A}$ made to the oracle. The adversary eventually outputs (m, t).*
3. *$\mathcal{A}$ wins if and only if*
   (a) $\mathsf{MacVer}_k(m, t) = 1$, *and*
   (b) $m \notin Q$.
   In that case the experiment outputs 1. Otherwise, the experiment outputs 0.

The advantage of the adversary $\mathcal{A}$ in the $\mathsf{EUF\text{-}CMA}$ *experiment is defined to be:*

$$\mathsf{Adv}_{\mathsf{MAC}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) = \Pr\left[\mathsf{G}_{\mathsf{MAC}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) = 1\right].$$

Definition 5. *A* (digital) signature scheme *is a tuple of three PTT-algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{SigVer})$ *such that:*

1. *The* key generation algorithm Gen *takes the security parameter* 1^n *and outputs a public-/private-key pair (sk, pk): $(sk, pk) \leftarrow$* $\mathsf{Gen}(1^n)$.
2. *The* signing algorithm Sign *takes as input a private key sk and a message m from some message space (that may depend on pk). It outputs the signature σ and we write this as $\sigma \leftarrow$* $\mathsf{Sign}_{sk}(m)$.

3. *The deterministic* verification algorithm SigVer *takes as input a public key* pk*, a message* m *and a signature* σ*. It outputs a bit* b*, with* $b = 1$ *meaning* valid *and* $b = 0$ *meaning* invalid*. We write this as* $b := \mathsf{SigVer}_{pk}(m, \sigma)$*.*

We require correctness *from the scheme: If* $(sk, pk) \leftarrow \mathsf{Gen}(1^n)$ *then, except with negligible probability,* $\mathsf{SigVer}_{pk}(m, \mathsf{Sign}_{sk}(m)) = 1$*.*

For brevity we often denote the key generation algorithms from the various primitives as $\mathsf{Gen}()$, omitting the security parameter.

2.2 Canetti–Krawczyk Modular Design

The *modular approach*, arising originally in a 1998 paper of Bellare, Canetti and Krawczyk [3], is to first define protocols secure against a limited adversary which can then be promoted to protocols secure against a realistic adversary using a generic compiler. In the *authenticated links model* (AM) the adversary is not permitted to fabricate messages, but can otherwise control the network and deliver messages out of order or to different parties from those intended. Compilers, or *authenticators*, can be applied to messages in an AM protocol to obtain protocols in the *unauthenticated links model* (UM) where the adversary can alter or fabricate messages limited only by the available computational power.

In both the UM and AM, adversaries control the execution of protocols by initiating parties and then invoking parties with available queries, including with message inputs. (In Sect. 2.4 we describe the available adversarial queries.) Parties respond to input messages by following the protocol definition and to other queries as defined by the query. Each party computes *local output* which is available to the adversary. The local output includes protocol decisions, such as whether a message is accepted (see Sect. 2.4 for details).

Bellare et al. [3] provide a theorem showing that a secure protocol, $\varPi_{\mathsf{AM}}$, in the AM maps to a secure protocol in the UM, $\varPi_{\mathsf{UM}}$, if the mapping is defined by a valid authenticator. An authenticator is valid if an observer, or distinguisher, is unable to distinguish between the world where an adversary $\mathcal{A}$ is interacting with the $\varPi_{\mathsf{AM}}$ and the world where an adversary $\mathcal{U}$ is interacting with the protocol $\varPi_{\mathsf{UM}}$. This is captured in the notion of protocol emulation in Definition 7.

Definition 6. *The* AM-UM *distinguishing experiment,* $\mathsf{G}^{\mathsf{AM\text{-}UM\text{-}dist}}_{\varPi_{\mathsf{AM}}\text{-}\varPi_{\mathsf{UM}}}(\mathcal{D})$ *proceeds as follows: (1) A uniform bit* $b \in \{0, 1\}$ *is chosen. If* $b = 0$*,* $\mathcal{D}$ *will interact with* $\mathcal{A}$ *and the* AM *protocol* $\varPi_{\mathsf{AM}}$*. Otherwise, if* $b = 1$*,* $\mathcal{D}$ *will interact with* $\mathcal{U}$ *and the* UM *protocol* $\varPi_{\mathsf{UM}}$*. (2) To conclude the experiment,* $\mathcal{D}$ *will halt and output* b'*. (3) The experiment will output* 1 *if and only if* $b = b'$*. We define the advantage of the distinguisher* $\mathcal{D}$ *to be*

$$\mathsf{Adv}^{\mathsf{AM\text{-}UM\text{-}dist}}_{\varPi_{\mathsf{AM}}\text{-}\varPi_{\mathsf{UM}}}(\mathcal{D}) = 2 \cdot \left| \Pr\left[\mathsf{G}^{\mathsf{AM\text{-}UM\text{-}dist}}_{\varPi_{\mathsf{AM}}\text{-}\varPi_{\mathsf{UM}}}(\mathcal{D}) = 1\right] - \frac{1}{2} \right| .$$

Definition 7. *Let* $\varPi_{\mathsf{UM}}$ *and* $\varPi_{\mathsf{AM}}$ *be protocols in the* UM *and* AM *models respectively. We say that* $\varPi_{\mathsf{UM}}$ ϵ*-emulates* $\varPi_{\mathsf{AM}}$ *in unauthenticated networks if for any*

UM-adversary $\mathcal{U}$ interacting with $\mathit{\Pi}_{\mathsf{UM}}$, there exists an AM-adversary $\mathcal{A}$ interacting with $\mathit{\Pi}_{\mathsf{AM}}$ such that for any distinguisher $\mathcal{D}$ playing the AM-UM distinguishing game, $\mathsf{Adv}^{\mathsf{AM\text{-}UM\text{-}dist}}_{\mathit{\Pi}_{\mathsf{AM}}\text{-}\mathit{\Pi}_{\mathsf{UM}}}(\mathcal{D}) \leq \epsilon$.

An authenticator is a specific type of *protocol compiler* transforming one protocol into another. The modularity of the approach relies on the observation that an authenticator will actually preserve protocol security as we will see in Sect. 2.4.

Definition 8 ([3]). *An authenticator is a compiler, C, that takes an AM protocol $\mathit{\Pi}_{\mathsf{AM}}$ as input and outputs a UM protocol $\mathit{\Pi}_{\mathsf{UM}}$, such that $\mathit{\Pi}_{\mathsf{UM}}$ emulates $\mathit{\Pi}_{\mathsf{AM}}$.*

2.3 MT-authenticators

Defining an authenticator for any protocol, regardless of the number of messages, seems at first a difficult problem. To deal with this, BCK98 [3] define a simpler notion of an MT-authenticator, designed to authenticate a single arbitrary message. They also showed that repeated use of a valid MT-authenticator is a valid authenticator, so that protocol messages can be treated separately.

A bit more formally we define MT as a *message transmission* protocol in authenticated networks that works as follows: when P_i is activated with (P_j, m), party P_i sends the message $(\mathsf{P}_i, \mathsf{P}_j, m)$ to party P_j and outputs "P_i sent m to P_j". Upon receiving $(\mathsf{P}_i, \mathsf{P}_j, m)$, P_j outputs "P_j received m from P_i". Note that the quoted outputs are local outputs of the parties and are critical in proving proper emulation; however, when we later show compiled protocols we omit mention of these local outputs.

An MT-authenticator, λ, is a protocol that emulates MT in unauthenticated networks. Given a sequence of MT-authenticators, $\mathit{\Lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_t)$, the derived protocol compiler, $C_{\mathit{\Lambda}}$, uses the next MT-authenticator to authenticate the next message. More precisely, given a protocol $\mathit{\Pi}$ in the AM with t messages, $m_1, m_2, \ldots, m_t$ the protocol $\mathit{\Pi}' = C_{\mathit{\Lambda}}(\mathit{\Pi})$ in the UM is defined as follows. For each message, m_k, sent in $\mathit{\Pi}$, λ_k is run to send the same message from the same initiator to the same recipient. Whenever a party, P_j, outputs "P_j received m from P_i" in λ_k, then $\mathit{\Pi}$ is activated at P_j with message m_k from P_i. If $\mathit{\Lambda}$ is a sequence of t MT-authenticators then $C_{\mathit{\Lambda}}$ is an authenticator. We restate this in Theorem 1 and give a sketch of the proof, which is given in full in [3].

Theorem 1 ([3]). *Let $\mathit{\Lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_t)$ be a sequence of t MT-authenticators so that each λ_k ϵ-emulates MT. Then the compiler, $C_{\mathit{\Lambda}}$, will be an authenticator such that for any protocol $\mathit{\Pi}$ in the AM, $C_{\mathit{\Lambda}}(\mathit{\Pi})$ $(t \cdot \epsilon)$-emulates $\mathit{\Pi}$ in the UM.*

Proof. Let $\mathit{\Pi}$ be an AM protocol. Let $\mathcal{U}$ be a UM-adversary interacting with $C_{\mathit{\Lambda}}(\mathit{\Pi})$. $\mathcal{A}$ runs $\mathcal{U}$ on a simulated interaction. Action requests from $\mathcal{U}$ to parties in the UM can be mimiced by $\mathcal{A}$ in the AM and $\mathcal{A}$ relays its results back to $\mathcal{U}$. The only problem with the simulation could occur in the case that $\mathcal{U}$ specifies that a message is received by some party P_j from some party P_i in the UM, but

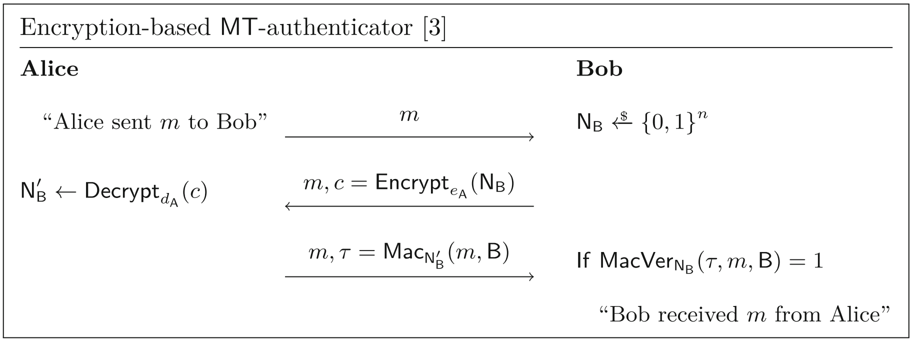

Encryption-based MT-authenticator [3]
Alice
Bob
"Alice sent m to Bob"
m
$\mathsf{N_B} \xleftarrow{\$} \{0,1\}^n$
$\mathsf{N'_B} \leftarrow \mathsf{Decrypt}_{d_\mathsf{A}}(c)$
$m, c = \mathsf{Encrypt}_{e_\mathsf{A}}(\mathsf{N_B})$
$m, \tau = \mathsf{Mac}_{\mathsf{N'_B}}(m, \mathsf{B})$
If $\mathsf{MacVer}_{\mathsf{N_B}}(\tau, m, \mathsf{B}) = 1$
"Bob received m from Alice"

Fig. 1. Bob authenticates message m from Alice.

that message is not in the set of messages waiting for delivery in the AM. But this can happen with probability bounded by ϵ. Such an event could occur for any of the t messages and so the probability that the simulation is correct is at least $(1-\epsilon)^t \geq 1 - t \cdot \epsilon$. Finally, any observer will be able to distinguish between the run of Π in the AM and $C_\Lambda(\Pi)$ in the UM with advantage at most $\epsilon' = t \cdot \epsilon$.

Note that although we have assumed that each MT-authenticator has the same security level ϵ, the theorem is still true if the MT-authenticators have different security levels $\epsilon_1, \epsilon_2, \ldots, \epsilon_t$ and we take $\epsilon = \max_k(\epsilon_k)$. In the cases we are interested in, we will always have $t = 2$.

An example of an MT-authenticator is the encryption-based authenticator [3] shown in Fig. 1, where N_B denotes a nonce and (e_A, d_A) an encryption-decryption keypair. This is a valid MT-authenticator as long as the public key encryption used is CCA-secure and the MAC is secure. The protocol can be optimized in various ways, as we will show later.

There exist several other MT-authenticators defined in the literature including signature-based [3], MAC-based [9] and password-based [17]. We show, in compressed form, the signature-based MT-authenticator later (Fig. 8). The modular approach allows combination of any MT-authenticator together with any AM-protocol, resulting in automatically secure UM protocols. Therefore adding any new building block, either an MT-authenticator or an AM-protocol, results in several new protocols of potential interest.

2.4 SK-security

SK-security is the AKE security notion of CK01 [9], capturing session key indistinguishability and correctness of the protocol. To define this notion we need to state the capabilities of the adversary and the indistinguishability experiment. Each protocol run at a party A is associated with a session identifier s. In the AM the value s is an input at the start of a run to the initiator party. Later we will see that session identifiers can be replaced by protocol messages as long as parties can verify that no incomplete sessions between the same parties have the same session identifier. The state of a session consists of the following information:

- status – whether or not the session is complete, aborted, or still in progress;
- any ephemeral key material needed to complete the protocol;
- the session key, sk, if the protocol is completed and has not expired.

The global state of a party may include long-term authentication keys $pk_{\mathsf{A}}, sk_{\mathsf{A}}$. As in most AKE models, we do not explicitly model distribution of long-term keys. Furthermore, we assume that long-term public keys are immediately available to any party that needs them. This may be too strong an assumption in some real-world protocols, such as TLS, and we remark further on this issue when we examine concrete protocols in Sect. 5.

Definition 9 (Matching sessions [9]**).** *The two sessions* $(\mathsf{A}, \mathsf{B}, s, role)$ *and* $(\mathsf{A}', \mathsf{B}', s', role')$ *are* matching *if* $\mathsf{A} = \mathsf{B}'$*,* $\mathsf{B} = \mathsf{A}'$ *and* $s = s'$*.*

The adversary may issue the following queries, subject to certain restrictions we will see later.

- $\mathsf{NewSession}(\mathsf{A}, \mathsf{B}, s, r)$: the adversary issues the NewSession query to party A, specifying its intended partner B, the session identifier[1] s, and the role r (initiator or responder) of A in the session. A will follow the protocol definition and may return an output message intended for B.
- $\mathsf{Send}(\mathsf{A}, \mathsf{B}, m)$: represents activation of A by an incoming message m (possibly including a session identifier) from party B. A will follow the protocol and may reject, accept, or return an output message intended for B.
- $\mathsf{Corrupt}(\mathsf{A})$: the adversary learns the whole state of A including any long-term keys. The corruption event is recorded in the local output of A. Subsequently A can never be activated but the adversary can take the role of A in the protocol.
- $\mathsf{RevealKey}(\mathsf{A}, \mathsf{B}, s)$: the adversary learns the session key accepted in the session s by A with partner B, if it exists. The reveal event is recorded in the local output of A.
- $\mathsf{RevealState}(\mathsf{A}, \mathsf{B}, s)$: the adversary learns the state information associated with session s at A, such as ephemeral keys. The reveal state event is recorded in the local output of A.
- $\mathsf{Expire}(\mathsf{A}, \mathsf{B}, s)$: if there is a completed session s at A with B then any session key associated with that session is deleted from the memory of A. The Expire event is recorded in the local output of A.
- $\mathsf{Test}(\mathsf{A}, \mathsf{B}, s)$: this query can be asked only once and can only be made to a completed session s at A with partner B. Furthermore there cannot have been any of the following queries made: $\mathsf{RevealKey}(\mathsf{A}, \mathsf{B}, s)$ or $\mathsf{RevealState}(\mathsf{A}, \mathsf{B}, s)$ or $\mathsf{Corrupt}(\mathsf{A})$ or $\mathsf{Corrupt}(\mathsf{B})$. If the bit b specified by the challenger is $b = 1$ then the session key is returned. Otherwise $b = 0$ and a random key from the keyspace are returned.

Now we are in a position to define the SK-security experiment.

[1] We remark that instantiation of session identifiers differs between the models. In UM, s can be blank as the session identifier need not be determined by the adversary.

Definition 10. *The key indistinguishability experiment,* $\mathsf{G}_{\Pi}^{\mathsf{Key\text{-}Ind}}(\mathcal{A})$ *is defined as follows: (1) The challenger chooses a random bit* b *needed to define the* Test *query response. (2) The challenger initialises* n *parties and any long-term keys. (3)* $\mathcal{A}$ *may issue queries as defined above. (4) Eventually* $\mathcal{A}$ *halts and outputs a bit* b' *to indicate its guess for* b*, based on the response to the* Test *query. The experiment outputs 1 if and only if* $b' = b$*.*

Definition 11. *A key exchange protocol* Π *is* $\epsilon - \mathsf{SK}$*-secure if the following holds for any adversary* $\mathcal{A}$*:*

- *two* honest *parties (i.e. uncorrupted parties who faithfully execute the protocol instructions) completing matching sessions of the protocol* Π *will output the same key, except with negligible probability, and*
- *the advantage of the adversary* $\mathcal{U}$ *in the key indistinguishability experiment is:* $\mathsf{Adv}_{\Pi}^{\mathsf{Key\text{-}Ind}}(\mathcal{A}) = 2 \cdot |\Pr[b' = b] - 1/2| \leq \epsilon$*.*

The final step needed to bring the modular approach together is to show that emulation preserves SK-security. This was proven in CK01 and we re-state and re-prove it as Theorem 2 including concrete bounds. Note that using emulation of an ideal key exchange process as a definition of security, the original idea of BCK98, results in too strong a definition to allow some well-known protocols to be proven secure [9, Appendix A].

Theorem 2 **([9]).** *Let* Π *be an* $\epsilon{-}\mathsf{SK}$*-secure protocol in the* AM *with* t *messages. Let* C_{Λ} *be the compiler based on* MT*-authenticators* $\lambda_1, \lambda_2, \ldots, \lambda_t$ *such that for any protocol* Π *in the* AM*,* $C_{\Lambda}(\Pi)$ α*-emulates* Π *in the* UM*. Then protocol* $\Pi' = C_{\Lambda}(\Pi)$ *is an* $\epsilon' - \mathsf{SK}$*-secure protocol in the* UM *with* $\epsilon' = \epsilon + \alpha$*.*

Proof. Assume to the contrary that there exists a UM adversary $\mathcal{U}$ that has advantage ϵ' in the UM. Using $\mathcal{U}$, we build an AM adversary, $\mathcal{A}$, playing the game of Definition 10. When $\mathcal{A}$ receives its setup information consisting of system parameters and public keys from its challenger, $\mathcal{A}$ sends the same information to $\mathcal{U}$. Then $\mathcal{A}$ invokes $\mathcal{U}$ and mimics its behaviour in the AM, using its challenger to respond to the action requests when any party is exposed.

When $\mathcal{U}$ sends a message to a party P_j from a party P_i in the UM, $\mathcal{A}$ sends the same message between the same parties in the AM. The emulation will be perfect unless $\mathcal{U}$ successfully sends a message m to some party P_j from P_i but m was never sent by P_i. In this case we will say that $\mathcal{U}$ made a forgery and we let forge be the event that a forgery happens at any time during the run of $\mathcal{U}$.

If forge occurs then $\mathcal{A}$ will abort the simulation and return a random bit to its challenger. Note that this also defines a distinguisher $\mathcal{D}$ which will always win in the case that forge occurs. If forge does not occur then at some point $\mathcal{U}$ will ask its Test query for a session s. $\mathcal{A}$ then announces session s for its own Test query in the AM, receives a real or random key, and returns it to $\mathcal{U}$. Eventually $\mathcal{U}$ will halt and output its bit which $\mathcal{A}$ copies as its response. In this case, $\mathcal{A}$ wins whenever $\mathcal{U}$ wins.

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins}|\mathsf{forge}] \cdot \Pr[\mathsf{forge}] + \Pr[\mathcal{A} \text{ wins}|\neg\mathsf{forge}] \cdot \Pr[\neg\mathsf{forge}] \\ &= 1/2 \cdot \Pr[\mathsf{forge}] + \Pr[\mathcal{B} \text{ wins}] \cdot (1 - \Pr[\mathsf{forge}]) \\ &\geq \Pr[\mathcal{B} \text{ wins}] - 1/2 \cdot \Pr[\mathsf{forge}]\end{aligned}$$

We also implicitly defined a distinguisher, $\mathcal{D}$, which wins when forge occurs or wins with probability at least $1/2$ when forge does not occur: $\Pr[\mathcal{D} \text{ wins}] \geq \Pr[\mathsf{forge}]/2 + 1/2$. Putting this together we get:

$$\mathsf{Adv}_{\Pi'}^{\mathsf{Key\text{-}Ind}}(\mathcal{U}) \leq \mathsf{Adv}_{\Pi}^{\mathsf{Key\text{-}Ind}}(\mathcal{A}) + \mathsf{Adv}_{\Pi-\Pi'}^{\mathsf{AM\text{-}UM\text{-}dist}}(\mathcal{D}).$$

2.5 Optimising the UM Protocol

Simple application of an MT-authenticator to each message of an SK-secure AM protocol results in an SK-secure UM protocol as proven in Theorem 2. However, such a protocol is far from optimal. The most obvious drawback is that a two-message protocol, such as Diffie–Hellman, compiles to a six-message protocol. The obvious way to optimise such a protocol is to "piggyback" messages going in the same direction. The resulting protocol may be secure, but formally this process may break the security proof because it may alter the order of the local output of the parties, allowing trivial distinguishability outputs of the AM protocol from the outputs of the compiled UM protocol [17].

Because of such issues, the modular approach of CK01 has been criticised [12] for not achieving efficient protocols. There is some truth in such criticisms—for example, when using signature- or encryption-based authenticators it is not possible to achieve secure 2-message AKE protocols which are often seen in the literature. Fortunately, rigorous optimisations are not difficult to achieve, typically resulting in 3-message protocols as efficient as real-world protocols. Indeed, 3-message AKE protocols are necessary in any case to achieve desirable security properties such as mutual entity authentication or key confirmation.

Hitchcock et al. [17] designed a general technique for altering message ordering in a security-preserving way. This involved defining an intermediate model between the AM and UM, which they call the *hybrid model*. Rather than use this more comprehensive approach, here we apply simple techniques to allow optimisation of the number of messages and re-use message components as session identifiers. Consequently, the drawbacks of practical application of authenticators are removed resulting in generic protocols as efficient as standalone protocols.

Compressed Authenticators. The first step is to compress the authenticator to remove redundant elements. Notice that use of the authenticator in Fig. 1 expands each message m from the AM into three messages in the UM. However, sending m in all three messages is not actually needed (to achieve security), so we can simplify the encryption-based authenticator into a compressed version shown in Fig. 2. It is not hard to see [3,17] that removal of the repeated m fields does not affect the security of the MT-authenticator. Depending on the

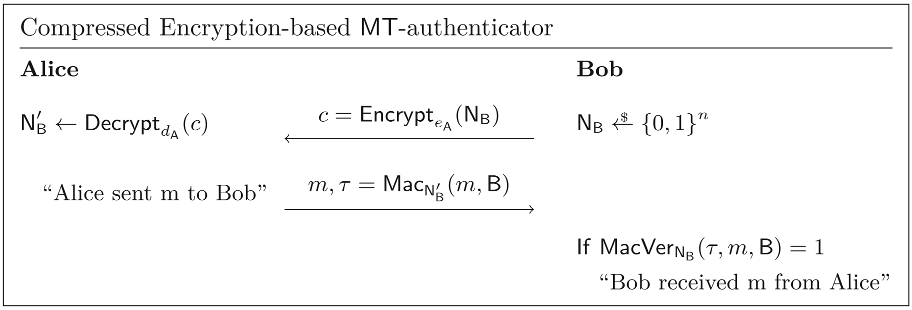

Compressed Encryption-based MT-authenticator
Alice
Bob
$\mathsf{N}'_{\mathsf{B}} \leftarrow \mathsf{Decrypt}_{d_{\mathsf{A}}}(c)$
$c = \mathsf{Encrypt}_{e_{\mathsf{A}}}(\mathsf{N}_{\mathsf{B}})$
$\mathsf{N}_{\mathsf{B}} \xleftarrow{\$} \{0,1\}^n$
"Alice sent m to Bob"
$m, \tau = \mathsf{Mac}_{\mathsf{N}'_{\mathsf{B}}}(m, \mathsf{B})$
If $\mathsf{MacVer}_{\mathsf{N}_{\mathsf{B}}}(\tau, m, \mathsf{B}) = 1$
"Bob received m from Alice"

Fig. 2. Compressed version of MT-authenticator in Fig. 1.

application scenario, the version in Fig. 1 may remain appropriate. The version in Fig. 2 is useful in a situation where Bob knows that some message, as yet unknown, will be authentically received from Alice; this case typically occurs in AKE protocols. Later we will see that to apply optimisation it is important that the first message in Fig. 2 is independent of the message to be authenticated, so that it can be generated and sent early in the protocol.

Session Identifiers. In the original formulation of CK01, session identifiers are sent in each protocol run in the AM. These must be unique for each active protocol run between the same parties, but it is not defined how they should be obtained in practice. Although the only property required of session identifiers is uniqueness, a natural way of obtaining them is to use random values chosen by each party; in that case the probability that session identifiers are not unique is negligible. In practice it may not be a burden for each party to ensure that there are no other incomplete sessions with the same identifier so that uniqueness is unconditionally guaranteed.

We assume that higher communication layers will provide a mechanism to ensure that messages get delivered to the correct session. They can also be explicitly added to the protocol messages if desired.

3 KEM-Based Building Blocks

This section defines and proves security for the basic KEM-based MT-authenticator and AM protocol, which will be brought together in Sect. 4 as components in defining generic efficient KEM-based AKE.

3.1 KEM-Based MT-authenticator

Figure 3 illustrates our KEM-based MT-authenticator. The construction is closely related to the encryption-based authenticator of BCK98.

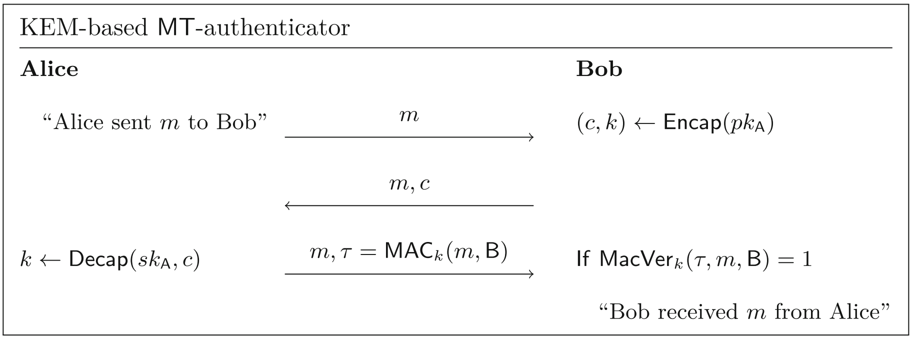

KEM-based MT-authenticator
Alice
Bob
"Alice sent m to Bob"
m
$(c, k) \leftarrow \mathsf{Encap}(pk_{\mathsf{A}})$
m, c
$k \leftarrow \mathsf{Decap}(sk_{\mathsf{A}}, c)$
$m, \tau = \mathsf{MAC}_k(m, \mathsf{B})$
If $\mathsf{MacVer}_k(\tau, m, \mathsf{B}) = 1$
"Bob received m from Alice"

Fig. 3. KEM-based MT-authenticator, λ_{KEM}: Bob authenticates m from Alice.

Next we give a theorem that λ_{KEM} is secure, meaning that it emulates MT in unauthenticated networks, as long as the KEM used achieves CCA security. The proof of Theorem 3 follows the proof strategy of Bellare et al. [3, Prop. 5] for their encryption-based authenticator. The complete proof is given in the full version [8].

Theorem 3. *The* KEM*-based* MT*-authenticator,* λ_{KEM}*, in Fig. 3, when instantiated with a* CCA*-secure* KEM *and a secure* MAC *scheme,* ϵ*-emulates protocol* MT *in unauthenticated networks such that* $\epsilon \leq l \cdot (\mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{KEM}}(\mathcal{D}) + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC}}(\mathcal{F}))$ *where* $l = n_P^2 \times n_M$*,* n_P *is the number of parties that run the protocol and* n_M *is the maximum number of challenge messages that can be sent by any party.*

Now that λ_{KEM} is proven to be an MT-authenticator we can invoke Theorem 2 to conclude that λ_{KEM} can be used to authenticate messages in an SK-secure AM protocol and results in a SK-secure UM protocol. In order to optimise the resulting protocol we will want to use a compressed version of the authenticator (see Sect. 2.5) as shown in Fig. 4.

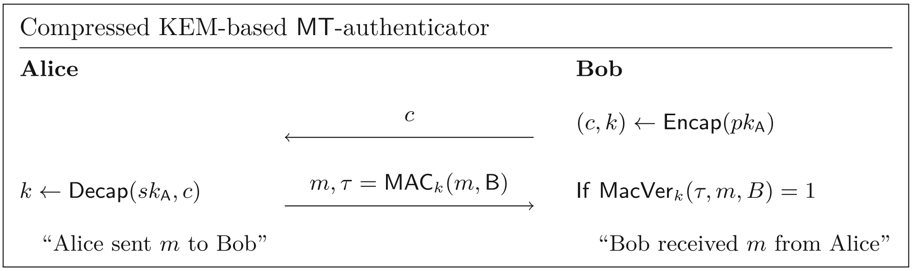

Compressed KEM-based MT-authenticator
Alice
Bob
c
$(c, k) \leftarrow \mathsf{Encap}(pk_{\mathsf{A}})$
$k \leftarrow \mathsf{Decap}(sk_{\mathsf{A}}, c)$
$m, \tau = \mathsf{MAC}_k(m, \mathsf{B})$
If $\mathsf{MacVer}_k(\tau, m, B) = 1$
"Alice sent m to Bob"
"Bob received m from Alice"

Fig. 4. λ_{KEM}, the compressed KEM-based MT-authenticator.

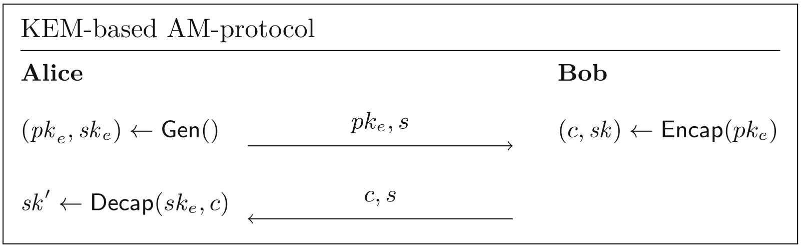

KEM-based AM-protocol
Alice
Bob
$(pk_e, sk_e) \leftarrow \mathsf{Gen}()$
pk_e, s
$(c, sk) \leftarrow \mathsf{Encap}(pk_e)$
$sk' \leftarrow \mathsf{Decap}(sk_e, c)$
c, s

Fig. 5. KEM-based protocol with any CPA-secure KEM (see Definition 2).

The security proof for the compressed authenticator is identical to the proof for the full authenticator since the only difference is the deletion of plaintext messages in the UM which are ignored in the security proof.

Corollary 1. *Theorem 3 still holds if the authenticator in Fig. 3 is replaced by the compressed* KEM*-based* MT*-authenticator,* λ_{KEM}*, in Fig. 4.*

3.2 KEM-Based AM Protocol

In Fig. 5 we present a KEM-based protocol Π that is SK-secure in the AM. The protocol is a generalisation of the basic Diffie–Hellman AM protocol of CK01 [9]. We assume that a setup with parameters for the KEM is known already to all parties. The initiator A will be invoked by the $\mathsf{NewSession}(\mathsf{A}, \mathsf{B}, s, r)$ query and responds with a new ephemeral KEM public key pk_e. Upon receipt of (pk_e, s) the responder encapsulates a new session key sk in c, and returns it to party A.

Theorem 4. *Let $\mathcal{A}$ be an adversary against the* SK*-security of protocol Π shown in Fig. 5. Let $\mathcal{A}$ interact with at most q sessions of Π for each pair of parties. Let n be the maximum number of parties involved in the protocol run. Then the advantage of $\mathcal{A}$ can be bounded by:* $\mathsf{Adv}^{\mathsf{SK}}_{\Pi}(\mathcal{A}) \leq n^2 q \cdot \mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{KEM}}(\mathcal{B})$.

The proof of Theorem 4 is in the full version [8].

4 Generic KEM-Based AKE Protocols

With the building blocks from Sect. 3 we now apply MT-authenticators to AM protocols and optimise them to obtain protocols which are both SK-secure in the realistic UM security model and efficient in comparison with other protocols in the literature. There is no restriction to apply the new MT-authenticator in Fig. 4 only to the new AM protocol in Fig. 5; the authenticator can be applied to any SK-secure AM protocol and any authenticator can be applied to the KEM-based AM protocol. Furthermore, we may apply different MT-authenticators to each of the messages in an AM protocol [17, Theorem 6] resulting in yet more ways to construct different secure protocols.

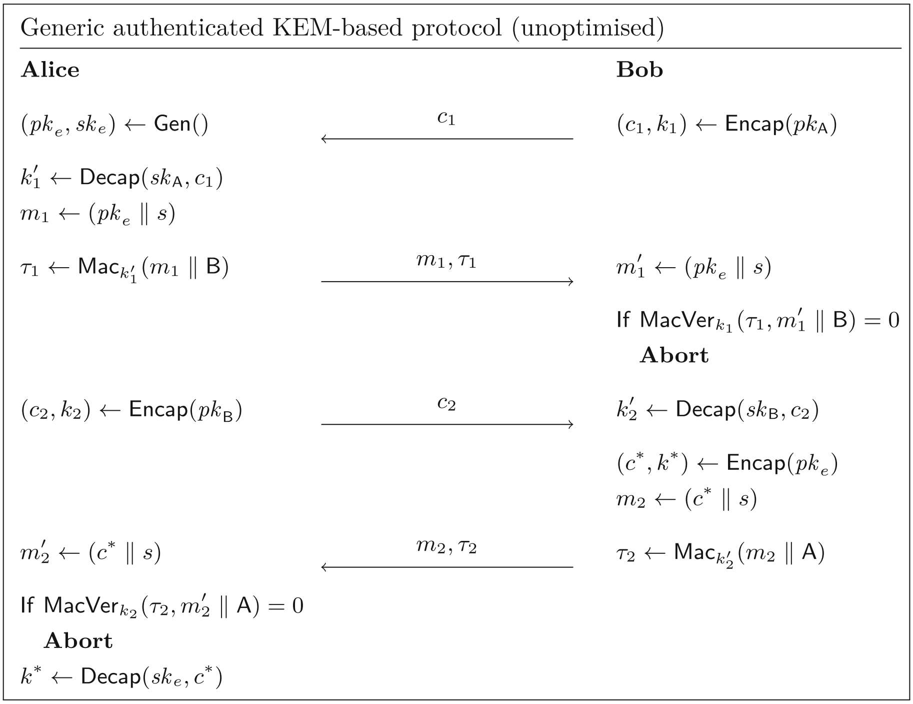
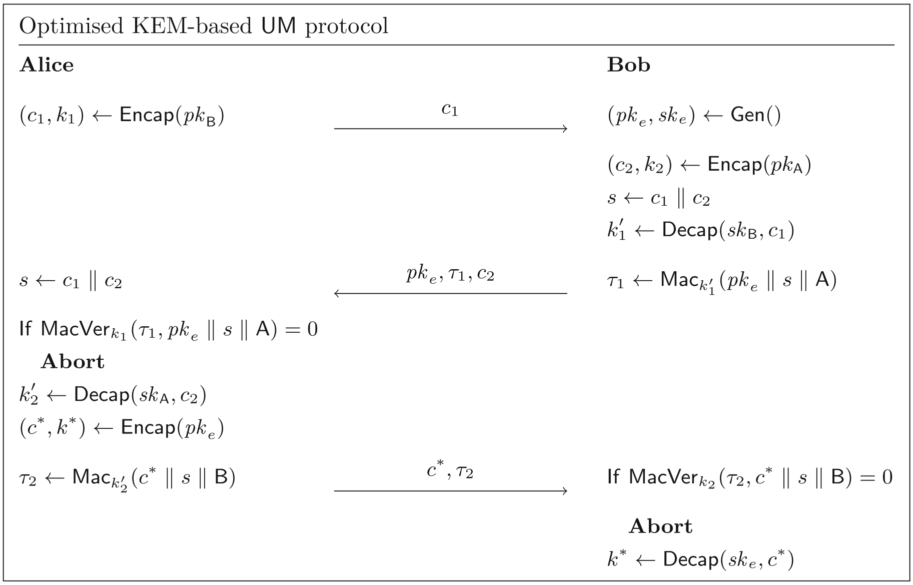

Generic authenticated KEM-based protocol (unoptimised)
Alice
Bob
$(pk_e, sk_e) \leftarrow \mathsf{Gen}()$
c_1
$(c_1, k_1) \leftarrow \mathsf{Encap}(pk_\mathsf{A})$
$k_1' \leftarrow \mathsf{Decap}(sk_\mathsf{A}, c_1)$
$m_1 \leftarrow (pk_e \parallel s)$
$\tau_1 \leftarrow \mathsf{Mac}_{k_1'}(m_1 \parallel \mathsf{B})$
m_1, τ_1
$m_1' \leftarrow (pk_e \parallel s)$
If $\mathsf{MacVer}_{k_1}(\tau_1, m_1' \parallel \mathsf{B}) = 0$
Abort
$(c_2, k_2) \leftarrow \mathsf{Encap}(pk_\mathsf{B})$
c_2
$k_2' \leftarrow \mathsf{Decap}(sk_\mathsf{B}, c_2)$
$(c^*, k^*) \leftarrow \mathsf{Encap}(pk_e)$
$m_2 \leftarrow (c^* \parallel s)$
$m_2' \leftarrow (c^* \parallel s)$
m_2, τ_2
$\tau_2 \leftarrow \mathsf{Mac}_{k_2'}(m_2 \parallel \mathsf{A})$
If $\mathsf{MacVer}_{k_2}(\tau_2, m_2' \parallel \mathsf{A}) = 0$
Abort
$k^* \leftarrow \mathsf{Decap}(sk_e, c^*)$

Fig. 6. Generic 4-message protocol obtained by compiling the KEM-based AM protocol with the compressed KEM-based MT-authenticator.

Due to our field's focus on post-quantum security in recent years, we emphasise KEM-based and signature-based components in this section, allowing us to apply any of the primitives from the NIST competition library. We illustrate this usage with several different examples in this section, applying both our new KEM-based authenticator and the existing signature-based authenticator to achieve a variety of protocols. Another example, also with potential for post-quantum security, is to apply the MAC-based authenticator of CK01 to our KEM-based AM protocol. This results in a protocol suitable for pre-shared key environments which is a common scenario, for example in TLS and IPSec. Details of a MAC-based generic protocol construction are available in Appendix 3.

4.1 Compiled KEM-Based Protocol and Optimization

We start with the AM-secure protocol from Fig. 5 and then apply the compiler consisting of application of the compressed MT-authenticator to each of its two messages. This leads to the 4-message protocol of Fig. 6.



Optimised KEM-based UM protocol

Alice

$(c_1, k_1) \leftarrow \mathsf{Encap}(pk_\mathsf{B})$

$s \leftarrow c_1 \parallel c_2$

If $\mathsf{MacVer}_{k_1}(\tau_1, pk_e \parallel s \parallel \mathsf{A}) = 0$

Abort

$k_2' \leftarrow \mathsf{Decap}(sk_\mathsf{A}, c_2)$

$(c^*, k^*) \leftarrow \mathsf{Encap}(pk_e)$

$\tau_2 \leftarrow \mathsf{Mac}_{k_2'}(c^* \parallel s \parallel \mathsf{B})$

$\xrightarrow{c_1}$

$\xleftarrow{pk_e, \tau_1, c_2}$

$\xrightarrow{c^*, \tau_2}$

Bob

$(pk_e, sk_e) \leftarrow \mathsf{Gen}()$

$(c_2, k_2) \leftarrow \mathsf{Encap}(pk_\mathsf{A})$

$s \leftarrow c_1 \parallel c_2$

$k_1' \leftarrow \mathsf{Decap}(sk_\mathsf{B}, c_1)$

$\tau_1 \leftarrow \mathsf{Mac}_{k_1'}(pk_e \parallel s \parallel \mathsf{A})$

If $\mathsf{MacVer}_{k_2}(\tau_2, c^* \parallel s \parallel \mathsf{B}) = 0$

Abort

$k^* \leftarrow \mathsf{Decap}(sk_e, c^*)$

Fig. 7. Optimised UM protocol from the KEM-based AM protocol and the KEM-based MT-authenticator.

Messages 1 and 2 are the result of applying the compressed MT-authenticator to authenticate the ephemeral public key pk_e generated by Alice. Messages 3 and 4 are the result of applying the compressed MT-authenticator to authenticate the encapsulated shared key c^* generated by Bob. The difference between m_1 and m_1' (resp. m_2 and m_2') in Fig. 6 is that both players have their own version of s—the MAC verifies the integrity of both the message and the session.

To optimise the 4-message protocol in Fig. 6 we take four simple steps: (1) The messages that were numbered 2 and 3 will be sent in parallel as a new message with number 2. This does not change the order or contents of any messages. (2) The session identifier, s, will be constructed by the parties as part of the protocol, instead of taking it as an external input to the protocol. Recall that the only requirements on s are to be unique between the parties amongst any incomplete protocol session between the two parties. We choose $s = c_1 \parallel c_2$ where c_1 and c_2 are the (randomised) encapsulations (ciphertexts) generated by each party. (3) Repeated message fields and fields previously generated by message receivers are removed from messages. (4) The protocol parties are relabelled so that Alice becomes the protocol initiator. Combining all of these steps we obtain the optimised protocol shown in Fig. 7.

As far as we are aware, the precise protocol of Fig. 7 is new in the literature. There are several existing protocols also aimed at achieving AKE based only on KEMs [15,18,23] or encryption [12]. Several of these are motivated by the desire to avoid signatures, which tend to suffer efficiency disadvantages compared with KEMs in the post-quantum examples from the NIST competition. The security

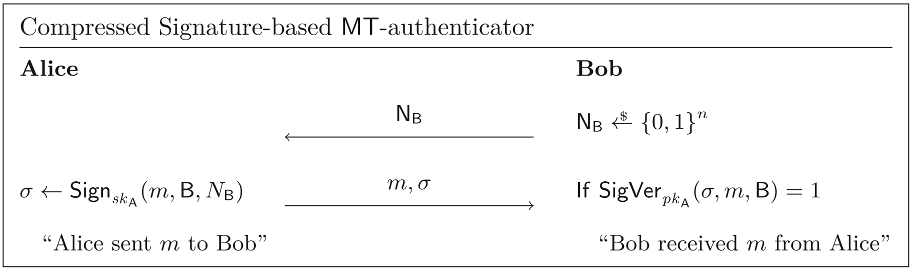

Compressed Signature-based MT-authenticator
Alice
Bob
$\mathsf{N_B}$
$\mathsf{N_B} \xleftarrow{\$} \{0,1\}^n$
$\sigma \leftarrow \mathsf{Sign}_{sk_\mathsf{A}}(m, \mathsf{B}, N_\mathsf{B})$
m, σ
If $\mathsf{SigVer}_{pk_\mathsf{A}}(\sigma, m, \mathsf{B}) = 1$
"Alice sent m to Bob"
"Bob received m from Alice"

Fig. 8. λ_{Sign}, a compressed signature-based MT-authenticator.

varies between of each these protocols. For example, the FSXY protocol [15] provides security against ephemeral key leakage whereas KEMTLS [23], like the protocol of Fig. 7, lacks this property. On the other hand, our protocol does allow state reveals from non-target sessions. KEMTLS is also designed to provide only one-way (server) authentication. Making a judgement on which of these protocols is "better" is therefore difficult since it depends on the security requirements and implementation details. In Sect. 5 we compare efficiency using concrete KEMs and signature schemes to get a better feel for the relative efficiencies.

4.2 Generic Protocols Using Signatures

We now look at two further generic protocols which we can obtain by using signatures in combination with our KEM-based AM protocol. We will need to apply the compressed signature-based authenticator shown in Fig. 8.

The authenticator λ_{Sign} is derived from the authenticator of BCK98 by removing the unnecessary message components in exactly the same way as for the encryption- and KEM-based authenticators. As before, the existing proof that the full authenticator is a valid MT-authenticator [3] still holds.

The optimised protocol for the KEM-based AM protocol compiled with the signature-based authenticator is shown in Fig. 9. The optimisation follows the same process as described in Sect. 4.1. Although more general, the resulting protocol has much in common with the signed Diffie–Hellman protocol which has been widely known and deployed for many years and is today the usual AKE in the latest version of TLS (though with only one-sided authentication).

We have another way to authenticate the KEM-based AM protocol, namely to authenticate its two messages with different MT-authenticators. As far as we are aware there are no examples of such a protocol in the existing literature. There can be practical usages, for example when signatures are very expensive to generate but very cheap to verify. In such a case, when a powerful server authenticates its AM message it can shift computation away from a lightweight client by using the signature-based authenticator, while the client can avoid generating signatures by using a different KEM-based authenticator. In Fig. 10 we show the optimised protocol using the KEM-based authenticator for the

Optimised KEM-based UM protocol with signature authentication

Alice		**Bob**
$\mathsf{N_A} \xleftarrow{\$} \{0,1\}^n$	$\xrightarrow{\mathsf{N_A}}$	$(pk_e, sk_e) \leftarrow \mathsf{Gen}()$
		$\mathsf{N_B} \xleftarrow{\$} \{0,1\}^n$ $s \leftarrow \mathsf{N_A} \parallel \mathsf{N_B}$
$s \leftarrow \mathsf{N_A} \parallel \mathsf{N_B}$	$\xleftarrow{pk_e, N_B, \sigma_1}$	$\sigma_1 \leftarrow \mathsf{Sign}_{sk_\mathsf{B}}(pk_e \parallel s \parallel \mathsf{A})$
If $\mathsf{SigVer}_{pk_\mathsf{B}}(\sigma_1, pk_e \parallel s \parallel \mathsf{A}) = 0$ **Abort** $(c^*, k^*) \leftarrow \mathsf{Encap}(pk_e)$		
$\sigma_2 \leftarrow \mathsf{Sign}_{sk_\mathsf{A}}(c^* \parallel s \parallel \mathsf{B})$	$\xrightarrow{c^*, \sigma_2}$	If $\mathsf{SigVer}_{pk_\mathsf{A}}(\sigma_2, c^* \parallel s \parallel \mathsf{B}) = 0$ **Abort** $k^* \leftarrow \mathsf{Decap}(sk_e, c^*)$

Fig. 9. Optimised UM protocol from the KEM-based AM protocol and the signature-based MT-authenticator.

first message and the signature-based authenticator for the second message. A mirror protocol results from using the two authenticators the other way around. For completeness this optimised protocol is given as Fig. 13 in Appendix 2.

5 Concrete Post-quantum Secure AKE Protocols

In the previous section we have presented optimised generic AKE protocols which will be secure as long as the KEM, signature and MAC primitives are instantiated with secure instances. Even restricting to a handful of currently best-trusted post-quantum primitives, this leads to hundreds of potential concrete protocols, bearing in mind that we have shown that different KEMs and signatures can be mixed in the same protocol and observing that the generic protocols are not symmetric between initiator and responder. The question of whether the concrete instantiated protocols are practical in terms of computational efficiency and message size is a natural one.

5.1 Computational Cost

To give an impression of the computational costs of our new protocols we summarize the number of public key operations needed in each of our optimised protocols in the upper part of Table 1. The lower part of the same table includes the number of similar operations for some prominent existing protocols.

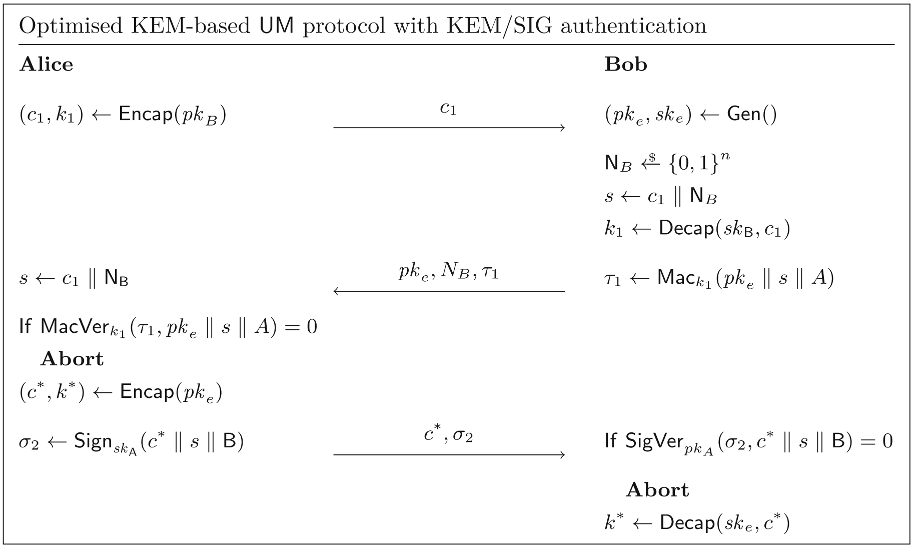

Optimised KEM-based UM protocol with KEM/SIG authentication
Alice
Bob
$(c_1, k_1) \leftarrow \mathsf{Encap}(pk_B)$
c_1
$(pk_e, sk_e) \leftarrow \mathsf{Gen}()$
$\mathsf{N}_B \xleftarrow{\$} \{0,1\}^n$
$s \leftarrow c_1 \parallel \mathsf{N}_B$
$k_1 \leftarrow \mathsf{Decap}(sk_\mathsf{B}, c_1)$
$s \leftarrow c_1 \parallel \mathsf{N}_\mathsf{B}$
pk_e, N_B, τ_1
$\tau_1 \leftarrow \mathsf{Mac}_{k_1}(pk_e \parallel s \parallel A)$
If $\mathsf{MacVer}_{k_1}(\tau_1, pk_e \parallel s \parallel A) = 0$
Abort
$(c^*, k^*) \leftarrow \mathsf{Encap}(pk_e)$
$\sigma_2 \leftarrow \mathsf{Sign}_{sk_\mathsf{A}}(c^* \parallel s \parallel \mathsf{B})$
c^*, σ_2
If $\mathsf{SigVer}_{pk_A}(\sigma_2, c^* \parallel s \parallel \mathsf{B}) = 0$
Abort
$k^* \leftarrow \mathsf{Decap}(sk_e, c^*)$

Fig. 10. Optimised UM protocol from the KEM-based AM protocol and the KEM-based MT-authenticator used for the first message, and the signature-based MT-authenticator for the second message.

Table 1. The number of public key operations for ours and existing protocols.

	Initiator					Responder				
	Gen_e	Encap	Decap	Sign	SigVer	Gen_e	Encap	Decap	Sign	SigVer
Figure 7. KEM/KEM	0	2	1	0	0	1	1	2	0	0
Figure 9. Sig/Sig	0	1	0	1	1	1	1	0	1	1
Figure 10. KEM/Sig	0	2	0	1	0	1	0	2	0	1
Figure 11. Sig/KEM	0	1	1	0	1	1	1	1	1	0
Figure 13. MAC/MAC	0	1	0	0	0	1	0	1	0	0
TLS 1.3[a]	1	0	1	0	1	0	1	0	1	0
KEMTLS[b] [23]	1	1	1	0	0	0	1	1	0	0
KEMTLS-pdk [24]	1	1	2	0	0	0	2	1	0	0
PQ-WireGuard [18]	1	1	2	0	0	0	2	1	0	0
SSW17[c] [12]	1	1[d]	2	0	0	0	2	1	0	0

[a] Using Diffie-Hellman as an ephemeral KEM. Unilateral authentication

[b] Unilateral authentication

[c] Assuming that our KEM-based AM protocol is used as the base protocol.

[d] Encryption is needed in the full protocol, not encapsulation

All of the protocols in Table 1 use three passes and three rounds. However, they do not all have the same goals or assumptions. TLS and KEMTLS only aim for server-side authentication while our protocol in Fig. 13 assumes pre-shared keys. PQ-WireGuard [18] is a variant of the WireGuard protocol using only KEMs. Its design is based on the FSXY protocol [15]. All of the protocols in the lower half of Table 1 use only KEMs, both for authentication and key exchange. When comparing with our KEM-only protocol of Fig. 7 we see that the main computational effort is the same as in the three bottom protocols which are all KEM-only protocols. We conclude that our protocols are comparable in computation to existing ones. Another difference between the various protocols is on which side most computations are performed, e.g. in Fig. 10 the initiator Alice encapsulates twice while Bob performs computationally heavier decapsulations and generation of the ephemeral key.

The most obvious difference between the upper and lower part of the table is that our designs have the responder generating the ephemeral KEM key while all existing protocols shown give this task to the initiating party. We do not believe that either option is inherently better, rather it depends on the relative costs of generation, encapsulation and decapsulation of the instantiating ephemeral KEM. For some well-known KEMs (Table 3a), key generation is far more costly than encapsulation or decapsulation. To minimise the overall protocol cost to both parties it seems better to use an algorithm with more uniform cost for the three KEM operations, but if it is desired to reduce the cost of one party at the expense of the other then different algorithms can be better.

It can be argued that implementation is most efficient when the same concrete KEM is used for all three of the KEM instances in the all-KEM protocols. This should be true at least with regard to the codebase needed in any implementation. However, this may not be the case when it comes to counting computation cycles. Recall that the AM protocol includes generation of an ephemeral public key, while the long-term keys are generated only once before the protocol runs. Therefore it can make sense to use a KEM with an efficient key generation algorithm for the ephemeral KEM, and a different one with a much less efficient key generation algorithm for the KEM using the long-term keys. PQ-WireGuard [18] does exactly this, using Classic McEliece for the long-term KEM and a variant of Saber for the ephemeral KEM. The size of its public key (Table 3a) shows why using Classic McEliece for the ephemeral KEM seems to be a bad idea.

Current known post-quantum signatures tend to be computationally less efficient than KEM constructions (Table 3b) where signing is much more expensive than decapsulation in known algorithms. It is therefore natural that KEM-based authentication currently is seen favourably. This can change in the future, and the NIST focus on new post-quantum signature proposals may well lead to more efficient post-quantum secure signature algorithms. To our knowledge, there is no analog to our KEM/Sig or Sig/KEM protocols in the literature, neither are we aware of post-quantum proposals for the pre-shared key case.

5.2 Communications Cost

In Table 2 we take an inventory of the message fields in each of our abstract protocols. Due to the optimisation techniques explained earlier, the number of fields sent and received by each party is three in all cases. Informally, at least, this is a minimum since the ephemeral public key needs to communicated and

Table 2. What comprises the messages sent in each protocol.

	Message 1	Message 2	Message 3
Figure 7. KEM/KEM	ct	pk, ct, MAC	ct, MAC
Figure 9. Sig/Sig	N	pk, N, σ	ct, σ
Figure 10. KEM/Sig	ct	pk, N, MAC	pk, MAC
Figure 11. Sig/KEM	N	pk, σ, ct	ct, MAC
Figure 13. MAC/MAC	N	pk, N, MAC	ct, MAC

then used in the response, and each of these two messages must be authenticated using a fresh value chosen by the other party.

The size of these fields depends on the parameters of the concrete primitives chosen. In July 2022, NIST announced a first list of selected candidates as a result of its Post-Quantum Cryptography competition [1], pointing out CRYSTALS-Kyber as their selected KEM and CRYSTALS-Dilithium as their selected signature algorithm. Using the real-world efficiency of the Kyber KEM and the Dilithium signature scheme, in Tables 3a and 3b and naively adding up these numbers, all messages in our Fig. 10 protocol would be under 5 kB for Kyber-1024, which definitely is practical. Another look at the ephemeral public key sizes in Table 3a shows why the choice of Saber in PQ-WireGuard [18] is an obvious one. We note that before its recent demise, SIKE looked an even more promising candidate to minimise the ephemeral key size.

Just as for computation efficiency, currently accepted post-quantum secure signature candidates do not look attractive for communications efficiency as shown in Table 3b. To minimise signature size FALCON is a better choice than Dilithium, but requires a trade-off with computation.

We reiterate that Table 2 assumes that authentic long-term public keys are available to all parties by some external channel. This fits some real-world protocols (such as WireGuard) but not others (such as TLS). Post-quantum signatures used to certify the long-term public keys can be chosen independently of other concrete choices in the protocol. This choice will obviously affect both the computation for each party and the size of the protocol messages. Although registration of public keys can avoid use of post-quantum signatures [16], it seems necessary to use signatures to achieve usual certificate properties.

6 Conclusion and Future Work

As summarized in Sect. 1.2, the main contributions of this work are the new KEM-based authenticator and corresponding security proof, the new proofs in the AM-model and the derivation of several new generic AKE protocols. We hope that the flexibility of protocol designs can be useful in fitting AKE protocols to different application use cases and that as new concrete KEMs and signature schemes are developed the generic protocols will yield new and interesting instantiations. Some of the ways to extend the work are the following.

- Adding additional security properties; for example, application of the twisted-PRF trick [15] can likely be added to an AM-protocol to secure against ephemeral leakage.
- Check whether use of hybid-KEMs (secure against conventional adversaries based on traditional assumptions and secure against post-quantum adversaries based on new assumptions) can be usefully applied to obtain hybrid-secure AKE [6].
- Since the modular approach does not naturally lead to tight reductions, it would be useful to improve on this, although in the end it may be necessary to complement new protocol designs obtained from the modular approach with a monolithic proof in a stronger model for some concrete protocol.
- Applying an authenticator just to one message (from a two-message AM protocol) will allow for unilateral authentication such as is common in TLS. The security and efficiency of such a generic protocol deserves analysis.
- Real-world implementations of the generic protocols is also left to future work—obviously this would be a prerequisite for future adaptation, and give us a more concrete comparison of their efficiency in the real world.

Appendix 1: ETSI Tables

Tables 3a and 3b present the computational efficiency of various KEMs and signatures from the NIST competition. The figures are taken from two recent reports from ETSI [25,26]. They are not intended as definitive efficiency comparisons—indeed some of the figures have already been improved upon—but rather to illustrate typical ballpark figures and highlight the big variation between many of the existing proposals.

We find it interesting to remark on a major difference regarding the *symmetry* of the computation requirements between Diffie–Hellman and the AM protocol (Fig. 5) which can be regarded as a generalisation. The computational requirements for Diffie–Hellman are the same for both initiator and responder. In the AM protocol the initiator runs Gen and Decap while the responder runs only Encap. Of itself this is not significant, since Encap really has two purposes: to generate the new key for the responder and to generate the encapsulation for the initiator. Thus in the Diffie–Hellman case the cost of Encap is the same as the cost of Gen plus the cost of Decap. However, in all the examples in Table 3a this is nowhere close to being true. Indeed Encap is always significantly cheaper than Gen plus Decap, which may be important when deciding which party take the role of initiator in a protocol run.

Table 3. Confusion matrix of one of the folds for the final classification of DGCNN-MS-T-W, with $W = 150$

(a) The efficiency of various KEMs [25].

	Gen	Encap	Decap	pk	ct	NIST security category
mceliece348864	36641040	44 350	134 745	261120	128	1
mceliece460896	117067765	117 782	271 694	524160	188	3
KYBER512	33856	45 200	59 088	800	768	1
KYBER1024	73544	97 324	115 332	1568	1568	3
ntruhps2048677	309216	83 519	59 729	930	930	1
ntruhps4096821	431667	98 809	75 384	1230	1230	3
LightSaber	45152	49 948	47 852	672	736	1
Saber	66727	79 064	76 612	992	1088	3

(b) The efficiency of various signature schemes [26].

	Sign	SigVer	σ	NIST security category
Dilithium-3	269 000	118 000	3293	3
Dilithium-5	429 000	179 000	4595	5
FALCON-512	386 678	82 340	666	1
FALCON-1025	789 564	168 498	1280	5

Appendix 2: Optimised KEM-Based UM Protocol with SIG/KEM Authentication

We give here in Fig. 11 an optimized protocol using the signature-based authenticator for the first message and the KEM-based authenticator for the second. This is reversed from the protocol in Fig. 10.

Appendix 3: MAC-Based MT-Authenticator

Canetti and Krawczyk [9] present also a MAC-based MT-authenticator (interestingly described only in compressed form) as shown in Fig. 12. This authenticator can be useful in scenarios where pre-shared keys exist such as in many use-cases of TLS with lightweight entities and also in session resumption in the latest TLS 1.3 version.

Since MACs are expected to remain secure in the post-quantum setting it makes sense to combine this authenticator with our KEM-based AM protocol to obtain a post-quantum secure AKE protocol suitable for pre-shared key applications. Figure 13 show the resulting optimised protocol.

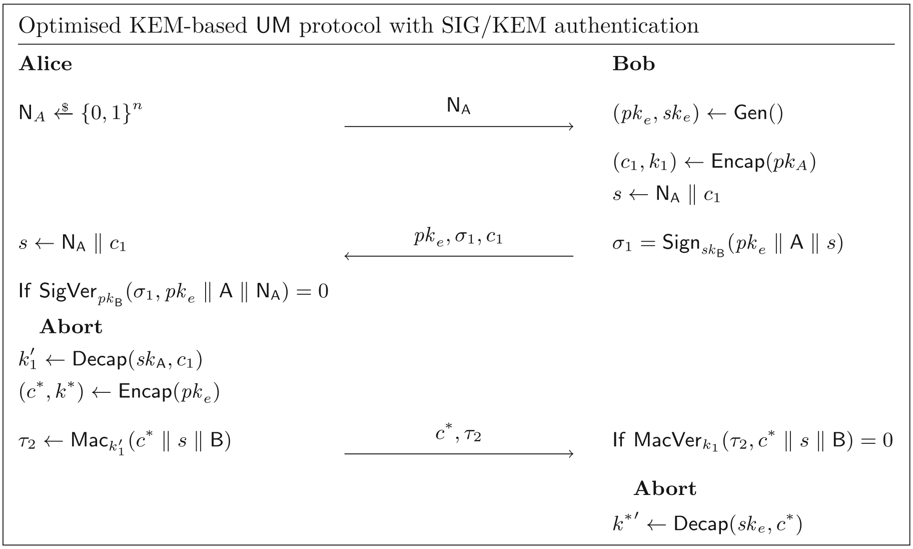

Optimised KEM-based UM protocol with SIG/KEM authentication
Alice
Bob
$\mathsf{N}_A \xleftarrow{\$} \{0,1\}^n$
N_A
$(pk_e, sk_e) \leftarrow \mathsf{Gen}()$
$(c_1, k_1) \leftarrow \mathsf{Encap}(pk_A)$
$s \leftarrow \mathsf{N}_\mathsf{A} \parallel c_1$
$s \leftarrow \mathsf{N}_\mathsf{A} \parallel c_1$
pk_e, σ_1, c_1
$\sigma_1 = \mathsf{Sign}_{sk_\mathsf{B}}(pk_e \parallel \mathsf{A} \parallel s)$
If $\mathsf{SigVer}_{pk_\mathsf{B}}(\sigma_1, pk_e \parallel \mathsf{A} \parallel \mathsf{N}_\mathsf{A}) = 0$
Abort
$k_1' \leftarrow \mathsf{Decap}(sk_\mathsf{A}, c_1)$
$(c^*, k^*) \leftarrow \mathsf{Encap}(pk_e)$
$\tau_2 \leftarrow \mathsf{Mac}_{k_1'}(c^* \parallel s \parallel \mathsf{B})$
c^*, τ_2
If $\mathsf{MacVer}_{k_1}(\tau_2, c^* \parallel s \parallel \mathsf{B}) = 0$
Abort
$k^{*\prime} \leftarrow \mathsf{Decap}(sk_e, c^*)$

Fig. 11. Optimised UM protocol from the KEM-based AM protocol and the signature-based MT-authenticator used for authenticating the first message, and the kem-based MT-authenticator authenticating the second message.

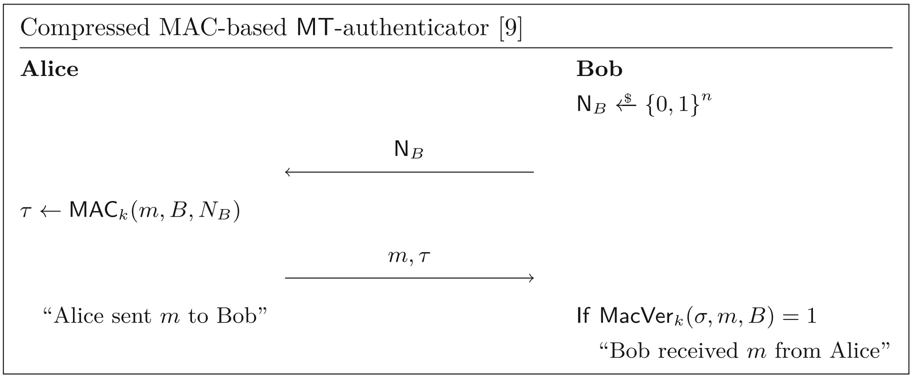

Compressed MAC-based MT-authenticator [9]
Alice
Bob
$\mathsf{N}_B \xleftarrow{\$} \{0,1\}^n$
N_B
$\tau \leftarrow \mathsf{MAC}_k(m, B, N_B)$
m, τ
"Alice sent m to Bob"
If $\mathsf{MacVer}_k(\sigma, m, B) = 1$
"Bob received m from Alice"

Fig. 12. λ_{MAC}, a compressed MAC-based MT-authenticator with shared key k.

Optimised KEM-based UM protocol with MAC authentication using pre-shared MAC key k

Alice		**Bob**
$\mathsf{N}_A \xleftarrow{\$} \{0,1\}^n$	$\xrightarrow{\mathsf{N}_A}$	$(pk_e, sk_e) \leftarrow \mathsf{Gen}()$
		$\mathsf{N}_B \xleftarrow{\$} \{0,1\}^n$ $s \leftarrow \mathsf{N}_A \parallel \mathsf{N}_B$
$s \leftarrow \mathsf{N}_A \parallel \mathsf{N}_B$	$\xleftarrow{pk_e, N_B, \tau_1}$	$\tau_1 \leftarrow \mathsf{MAC}_k(pk_e \parallel s \parallel A)$
If $\mathsf{MacVer}_k(\tau_1, pk_e \parallel s \parallel A) = 0$ **Abort** $(c^*, k^*) \leftarrow \mathsf{Encap}(pk_e)$		
$\tau_2 \leftarrow \mathsf{MAC}_k(c^* \parallel s \parallel B)$	$\xrightarrow{c^*, \tau_2}$	If $\mathsf{MacVer}_k(\tau_2, c^* \parallel s \parallel B) = 0$ **Abort** $k^* \leftarrow \mathsf{Decap}(sk_e, c^*)$

Fig. 13. Optimised UM protocol from the KEM-based AM protocol and the MAC-based MT-authenticator.

References

1. Alagic, G., et al.: Status report on the third round of the NIST post-quantum cryptography standardization process. Technical report, National Institute of Standards and Technology (2022). https://csrc.nist.gov/publications/detail/nistir/8413/final
2. Angel, Y., Dowling, B., Hülsing, A., Schwabe, P., Weber, F.: Post quantum noise. Cryptology ePrint Archive, Report 2022/539 (2022). https://eprint.iacr.org/2022/539
3. Bellare, M., Canetti, R., Krawczyk, H.: A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In: 30th ACM STOC, pp. 419–428. ACM Press (1998)
4. Bergsma, F., Jager, T., Schwenk, J.: One-round key exchange with strong security: an efficient and generic construction in the standard model. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 477–494. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_21
5. Bernstein, D.J., Lange, T.: Post-quantum cryptography. Nature **549**(7671), 188–194 (2017)
6. Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., Stebila, D.: Hybrid key encapsulation mechanisms and authenticated key exchange. In: Ding, J., Steinwandt, R. (eds.) PQCrypto 2019. LNCS, vol. 11505, pp. 206–226. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25510-7_12
7. Bos, J.W., et al.: Frodo: take off the ring! Practical, quantum-secure key exchange from LWE. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 1006–1018. ACM Press (2016)
8. Boyd, C., de Kock, B., Millerjord, L.: Modular design of KEM-based authenticated key exchange. Cryptology ePrint Archive, Paper 2023/167 (2023). https://eprint.iacr.org/2023/167

9. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_28
10. Celi, S., et al.: Implementing and measuring KEMTLS. Cryptology ePrint Archive, Report 2021/1019 (2021). https://eprint.iacr.org/2021/1019
11. Cremers, C.: Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In: Cheung, B.S.N., Hui, L.C.K., Sandhu, R.S., Wong, D.S. (eds.) ASIACCS 2011, pp. 80–91. ACM Press (2011)
12. de Saint Guilhem, C., Smart, N.P., Warinschi, B.: Generic forward-secure key agreement without signatures. In: Nguyen, P., Zhou, J. (eds.) ISC 2017. LNCS, vol. 10599, pp. 114–133. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69659-1_7
13. Ding, J., Alsayigh, S., Lancrenon, J., RV, S., Snook, M.: Provably secure password authenticated key exchange based on RLWE for the post-quantum world. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 183–204. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_11
14. Ding, J., Xie, X., Lin, X.: A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Paper 2012/688 (2012). https://eprint.iacr.org/2012/688
15. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly secure authenticated key exchange from factoring, codes, and lattices. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 467–484. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_28
16. Güneysu, T., Hodges, P., Land, G., Ounsworth, M., Stebila, D., Zaverucha, G.: Proof-of-possession for KEM certificates using verifiable generation. Cryptology ePrint Archive, Report 2022/703 (2022). https://eprint.iacr.org/2022/703
17. Hitchcock, Y., Boyd, C., Nieto, J.M.G.: Modular proofs for key exchange: rigorous optimizations in the Canetti-Krawczyk model. Appl. Algebra Eng. Commun. Comput. **16**(6), 405–438 (2006)
18. Hülsing, A., Ning, K.-C., Schwabe, P., Weber, F., Zimmermann, P.R.: Post-quantum WireGuard. Cryptology ePrint Archive, Report 2020/379 (2020). https://eprint.iacr.org/2020/379
19. Jager, T., Kiltz, E., Riepel, D., Schäge, S.: Tightly-secure authenticated key exchange, revisited. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 117–146. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_5
20. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, 2nd edn. CRC Press, Boca Raton (2014)
21. Krawczyk, H.: HMQV: a high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_33
22. Peikert, C.: A decade of lattice cryptography. Cryptology ePrint Archive, Paper 2015/939 (2015). https://eprint.iacr.org/2015/939
23. Schwabe, P., Stebila, D., Wiggers, T.: Post-quantum TLS without handshake signatures. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020, pp. 1461–1480. ACM Press (2020)

24. Schwabe, P., Stebila, D., Wiggers, T.: More efficient post-quantum KEMTLS with pre-distributed public keys. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021. LNCS, vol. 12972, pp. 3–22. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88418-5_1
25. ETSI Technical Committee Cyber Security. Quantum-safe public-key encryption and key encapsulation. ETSI TR 103823, ETSI (2021)
26. ETSI Technical Committee Cyber Security. Quantum-safe signatures. ETSI TR 103616, ETSI (2021)

Reusable, Instant and Private Payment Guarantees for Cryptocurrencies

Akash Madhusudan[1](✉), Mahdi Sedaghat[1], Samarth Tiwari[2], Kelong Cong[1], and Bart Preneel[1]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
{akash.madhusudan,ssedagha,kelong.cong,bart.preneel}@esat.kuleuven.be
[2] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
samarth.tiwari@cwi.nl

Abstract. Despite offering numerous advantages, public decentralized cryptocurrencies such as Bitcoin suffer from scalability issues such as high transaction latency and low throughput. The vast array of so-called Layer-2 solutions tackling the scalability problem focus on throughput, and consider latency as a secondary objective. However, in the context of retail payments, instant finality of transactions is arguably a more pressing concern, besides the overarching concern for privacy.

In this paper, we provide an overlay network that allows privacy-friendly low latency payments in a retail market. Our approach follows that of a recent work called Snappy, which achieved low latency but exposed identities of customers and their transaction histories. Our construction ensures this data is kept private, while providing merchants with protection against double-spending attacks. Although our system is still based upon customers registering with a collateral, crucially this collateral is reusable over time.

The technical novelty of our work comes from randomness-reusable threshold encryption (RRTE), a cryptographic primitive we designed specifically for the following features: our construction provably guarantees payments to merchants, preserves the secret identity of honest customers and prevents their transactions from being linked. We also present an implementation of our construction, showing its capacity for fast global payments in a retail setting with a delay of less than 1 s.

1 Introduction

Public decentralized cryptocurrencies such as Bitcoin and Ethereum offer increased transparency and avoid trust in a central party. However, these advantages come at the cost of performance, rendering them unfit for high throughput, real-time applications. The throughput of cryptocurrencies is orders of magnitude lower than that of traditional payment service providers such as Visa. Further, transactions require time before being considered final: the convention has been to wait for 6 confirmations or approximately an hour. For this reason, securely improving throughput and transaction confirmation latency (hereon referred to as latency) of public cryptocurrencies is a major area of research.

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 580–605, 2023.
https://doi.org/10.1007/978-3-031-35486-1_25

Layer-2 solutions tackle these constraints by offloading some elements of transactions off-chain [25]. However, these solutions focus on maximizing throughput and not on minimizing latency, which is dealt with as a secondary objective. Rollups, an increasingly popular solution in both industry and academia alike, increase the transaction throughput of Ethereum up to 100 times compared to its current throughput [7,11,20]. Yet, their latency matches that of the underlying blockchain. Payment Channel Networks (PCNs), another popular scaling solution, allow for arbitrarily many transactions on their network at the cost of a constant number of on-chain transactions, thereby drastically increasing throughput and reducing transaction fees. Yet, there are various factors negatively impacting the latency of PCN transactions, such as liveness of intermediate nodes [18], and the route discovery mechanisms [33]. Retail payments is a scenario where instant finality (hereon referred to as fast payments) is of the utmost importance; waiting an hour for a coffee is not an option. Additionally, retail payments are usually unilateral, i.e., from customers to merchants. An acknowledged problem with PCNs is channel depletion i.e., repeated use of channels in the same direction results in depleted channels, prohibiting further payments in the same direction [4]. Unilateral retail transactions only aggravate this issue. Similarly, by updating its layer-1, Ethereum 2.0 has increased its transaction throughput significantly. However, latency still takes $\sim$14 min [21]. This suggests a decoupling of two performance measures, namely throughput and latency, which need to be tackled separately.

Hence, *collateral reusability* and *fast payments* become interesting properties for solutions that perform retail payments with cryptocurrencies. Snappy is a fast on-chain payment system designed for a retail environment, where payers can assure payees of their ability to pay for a certain commodity [29]. Payees are protected from double-spending at rates much faster than the underlying blockchain while allowing collaterals to be reused. In order to work, Snappy depends on a set of *statekeepers* responsible for proactively detecting double-spending attempts in the system. These statekeepers have access to all transactions made by the customer, either by simply querying the blockchain or locally logging them when they receive it. Thus, their system suffers from privacy issues and all transactions involving the same payer can be linked together by third parties, such as, the statekeepers. The general demand for greater privacy becomes even more relevant in the retail context. For instance, the retail giant Target was able to deduce the pregnancy of a teenager even before her own parents found out [27]. Such unfortunate leaks can be prevented, if merchants are unable to link customers' various purchases. Hence, there is a need for a solution that has the following properties:

P1. Instant payments with double-spending protection that is much faster than the underlying blockchain (*fast payments*).

P2. Prevents third parties in the system from being able to link different transactions involving the same honest payer (*transaction unlinkability*).

P3. Privacy for honest customers is provided efficiently without incurring latency constraints of payments (*efficient privacy*).

P4. Honest payers do not need to replenish their collaterals (*reusability*).

At ICDCS'21, Ng et al. [31] proposed LDSP that adds privacy to Snappy. LDSP achieves **P1**, **P2** and **P3**; however, at the cost of reusable collaterals (**P4**). Thus, previous works either trade-off privacy in order to achieve fast payments with collateral reusability or vice-versa.

Our Contributions. In this paper, we provide a new overlay network that fulfills all aforementioned properties. To the best of our knowledge, we are the first to simultaneously achieve the combination of properties (**P1**–**4**) in a payment system utilised in a retail context. Our contributions may be split into three as follows:

Private Low Latency Double-Spending Protection. We provide an overlay network capable of providing payees protection from double-spent transactions much faster than the underlying blockchain. No party in the system is able to link different transactions involving the same honest payer, nor do these privacy guarantees adversely affect latency of transactions. Customer collaterals are also *reusable*, so that customers can repeatedly guarantee payments over time.

Formal Security Analysis. We formally prove that our construction preserves the secret identity of honest customers (anonymity) and disables anyone from linking their transactions (unlinkability), yet at the same time guarantees payment to the merchants (payment certainty).

Implementation and Evaluation. We implement our construction and show that it allows for fast global retail payments with a delay of less than 1 s.

Outline. The rest of this paper is organized as follows: Sect. 2 gives an overview of our construction, its participants and how they interact. Section 3 formally describes our construction, its threat model and proves how our construction satisfies its security requirements. Section 4 depicts an efficient instantiation of our construction and lists the cryptographic primitives used in detail. In Sect. 5 we evaluate the performance of this instantiation while focusing on latency. In Sect. 6 we discuss our limitations and point out the differences our construction has when compared to similar state-of-the-art research and finally in Sect. 7 we conclude our paper.

2 Our Construction

2.1 Our Solution

The strawman solution shown in the full version [28] depicts the difficulty of a straightforward solution in achieving **P1**–**4**.

While designing a solution that fulfills all aforementioned properties, we faced two challenges. First, our construction must utilize a proactive double-spending protection mechanism that is able to *selectively* reveal information of dishonest parties while still keeping all information about honest parties private. Second, since the payment latency needs to be low, we must enable payers to prove vital information privately yet very efficiently. We address the first challenge by proposing a novel variant of threshold encryptions, which we hereon refer to as randomness-reusable threshold encryptions (RRTE) that has the unique property of revealing the plaintext if two ciphertexts are encrypted using the same randomness. We use RRTE in combination with Pseudo-Random Functions (PRF) such that the statekeepers are able to proactively protect from double-spend attempts without having knowledge of any transaction details; yet, are still able to reveal vital information in the case of double-spent transactions. The second challenge is addressed by a combination of Threshold Structure-Preserving Signatures (TSPS) [16] and Non-Interactive Zero-Knowledge Proofs (NIZKs). The compatibility of TSPS with efficient NIZKs enables a payer to prove vital information to payees in our system in zero-knowledge, without impacting the latency of transactions.

Participants. First, in order to explain the interplay of these cryptographic primitives that solve the aforementioned challenges and fulfill **P1**–**4**, we introduce the participants of our construction. Similar to Snappy, our participants include: (1) customers willing to purchase products/services using their cryptocurrencies while expecting low latency; (2) an established consortium of merchants willing to accept cryptocurrency payments and (3) statekeepers who are selected from the merchant consortium. Additionally, we also include a group of authorities *trusted* for registration of users and verification of collaterals. Although authorities have greater power during the setup of our network, they *do not* have any access to future transactions between a customer and a merchant. For more discussion on these trust assumptions see Sect. 6.

Interaction of Participants. In our construction, the participants function as follows; each *customer* in our systems owns collateral(s), which is uniquely linked to a secret PRF key(s). This secret PRF key is signed by the group of *authorities* by utilizing TSPS once they verify its existence on the arbiter smart contract. The PRF is used in combination with our RRTE to set up our double-spending protection. By using this signed secret PRF key, each customer generates a randomness, which they then use in combination with the public key of *target merchant* to encrypt their secret identity. For encryption we use RRTE, that reveals the secret identity of a malicious customer (plaintext) when they double-spend. More precisely, double-spending in our system amounts to certifying multiple transactions with the same collateral, by which RRTE reveals the dishonest customer's identity. Thus, our construction fulfills **P1** by enabling any *statekeeper* who receives two ciphertexts that are encrypted using the same randomness to catch double-spending and reveal the identity of the perpetrator. However, honest customers who only certify one transaction per collateral remain unaffected; hence, also fulfilling **P2**.

Next, the customer needs to convince the target merchant about the existence of their collateral. However, they must do this without revealing any identifying details. To achieve this, we utilize TSPS and NIZKs. The payment guarantee is an indirect proof of collateral by proving knowledge of the TSPS provided to them by the authorities. In case of a double-spend attempt, this payment guarantee is sufficient for the merchant to be reimbursed. This can be done efficiently by efficient proof systems fulfilling **P3**.

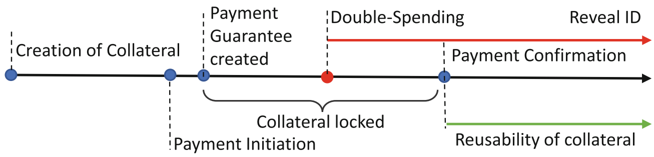


Fig. 1. Timeline of a transaction.

Finally, our construction also allows a customer to *reuse* their collaterals after a certain time interval in order to achieve **P4**. As illustrated in Fig. 1, a collateral is considered *locked* for the time period between the customer sending a payment guarantee to the merchant and the actual confirmation of their blockchain transaction. However, after the confirmation of a blockchain transaction, the customer's collateral can be reused. Note that this is similar to the execution of a multi-hop payment on PCNs, that rely on Hashed-Time-Locked-Contracts, locking up the collateral for a similar time period [1,33]. Thus, using a collateral twice in quick succession is treated as a double-spend and can be detected, but it may be reused once the locking period of the collateral has elapsed.

Construction Overview. Figure 2 illustrates our construction. In order to explain our construction, we assume an established set of authorities and a fixed merchant consortium. The payment protocol between a customer and a merchant has off-chain and on-chain components that have been depicted in Fig. 2 with dotted and solid arrows respectively. The protocol proceeds as follows:

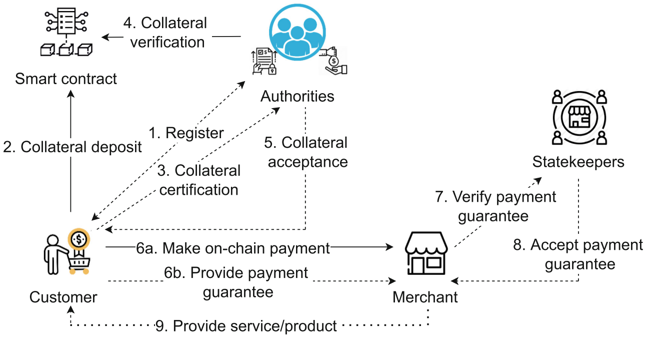

4. Collateral verification
Smart contract
Authorities
Statekeepers
2. Collateral deposit
1. Register
3. Collateral certification
5. Collateral acceptance
7. Verify payment guarantee
8. Accept payment guarantee
6a. Make on-chain payment
6b. Provide payment guarantee
Customer
Merchant
9. Provide service/product

Fig. 2. Our construction.

1) The customer begins by registering themselves with a set of authorities. **2)** The customer deposits a collateral in the smart contract, controlled by this set of authorities. **3)** Once the deposit is confirmed on the blockchain, the customer requests a collateral certification, which is necessary to generate payment guarantees for merchants. **4)** Upon receiving the certification request of the customer, the authorities check the smart contract to confirm if the customer deposited a collateral. **5)** The authorities provide the customer with a signed collateral certification. **6a)** During a transaction in the retail market, the customer makes a payment to the target merchant using the underlying cryptocurrency. This is done by sending a payment to the merchant through the smart contract. **6b)** The customer simultaneously generates an encrypted *payment guarantee* by using the collateral certification and sends this to the target merchant along with the transaction identifier of their cryptocurrency payment. **7)** The target merchant forwards this encrypted *payment guarantee* to the statekeepers who individually confirm that it is not a double-spend attempt. The statekeepers compare the received guarantee with each guarantee they received within a fixed time period. If none of these comparisons reveal the plaintext, i.e., secret identity of the customer, they are guaranteed about its uniqueness. Note that this verification can be done by utilizing our proposed RRTE and does not require knowing a customer's identity. **8)** Each statekeeper returns a signed payment guarantee to the merchant. **9)** If a majority of the statekeepers return a confirmation, the merchant aggregates these signatures and accepts the payment guarantee and provides the customer with necessary services/products.

Double-Spending. Double-spending in our construction means a scenario where the customer is malicious and wants to double-spend their payment guarantee, i.e., use the same collateral to promise payments to two distinct merchants. In this case, when the statekeepers receive these guarantees from two distinct merchants, they are able to combine them and reveal the secret identity of the cheating customer. Along with this identity, a secret to the customer's collateral is also revealed. This secret is used by the victim merchant for remuneration. This is depicted in Fig. 3.

More details about how RRTE enables statekeepers to reveal the secret identity of a malicious customer are given in its formal definition in Sect. 4.

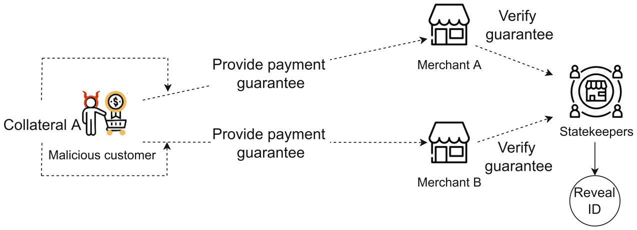


Fig. 3. Proactive double-spending detection.

Limitations. Although we address the privacy shortcomings of Snappy's design, our construction still inherits some of its other limitations, such as, the large collateral requirement for statekeepers and inadequately defined user incentives. To be specific, we also require the merchants to deposit collaterals in order to play the role of statekeepers fairly. We also assume that the amount of a customer's deposit is *fixed.* The assumption of a fixed set of merchants is also an additional concern, since it does not allow for any merchant churn in the protocol. However, in this paper we do not attempt to address these concerns. A more thorough discussion about these shortcomings is given in Sect. 6.

3 Preliminaries and Formal Construction

Throughout this paper, we let the security parameter of the scheme to be λ with unary representation of 1^λ, and $\mathsf{negl}(\lambda)$ denotes a negligible function. We use $x \leftarrow_\$ X$ to denote that x is sampled uniformly from the set X. $[n]$ denotes the set of integers in the range of 1 to n. For clarity, the secret values in our construction are represented with a *hat* operator (e.g., $\hat{\mathsf{sk}}$) and masked values are represented with the notation x' for the value x.

Threat Model. Our construction is designed to resist active adversaries that can corrupt a set of customers, merchants (which are also statekeepers) and authorities. To be precise, an adversary can only corrupt a minority of authorities during the initialization phase. During the processing of transactions, the adversary can corrupt all but one customers, and a minority of merchants.

Network Assumptions. We assume the existence of secure and reliable communication channels so that parties receive messages sent by honest parties eventually. The honest merchants/statekeepers are also assumed to be live and responsive, for the verification of transactions. We do not expect the customers to be online, unless they need to transact with a merchant. The underlying blockchain is assumed to be persistent and live and the adversary cannot influence the consensus mechanism.

Beyond the Threat Model. Let us mention a few considerations that are outside the scope of our model. Firstly, we consider protocol-level privacy, and not the privacy of the underlying blockchain. We do not consider side-channel attacks, such as metadata-level attacks on communication channels. Finally, the selection of authorities is also orthogonal to our work. One possibility is to choose them from the set of reputable merchants in the consortium, and have them shuffled periodically to ensure a majority of authorities always remain honest.

We would also like to note that weaker assumptions of trust or liveness are possible without compromising security. However, we select a simple set of assumptions, following those of Snappy, in order to focus on the privacy aspect instead of system-level optimizations.

Formal Construction. Our construction builds on Pseudo-Random Function (PRF), Non-Interactive Zero-Knowledge (NIZK) arguments, Digital Signatures (DS), Threshold Structure-Preserving Signatures (TSPS), Commitments (CO), and a novel randomness-reusable Threshold Encryption (RRTE). We list the formal definition and the security properties in the full version [28] and outline the scheme in Algorithm 1 for a relation $\mathbf{R_L}$ over three main NP-languages $\mathbf{L} := (\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3)$. The list of master public keys, mpk, is considered as an implicit input for all algorithms except the parameter generation algorithm (PGen). Additionally, all algorithms are PPT unless otherwise specified. We now formalise the functions in the Algorithm 1. All functions are split based on when they happen in our construction and are formalized as follows:

Bootstrapping Phase as Depicted in Fig. 2:

- $\mathsf{PGen}(1^\lambda, \mathbf{R_L})$ takes λ in its unary representation and relation $\mathbf{R_L}$ as inputs and returns the master public key mpk.
- $\mathsf{AuKeyGen}(\mathcal{AU})$ is executed by $\mathcal{AU}$ that returns $(\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai})$ for $i \in [n]$ along with a global verification key vk_a. $\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai}, \mathsf{vk}_a]_{i=1}^{n}$ represents the list of credentials for $\mathcal{AU}$.
- $\mathsf{MKeyGen}(\mathsf{M}_m)$ is executed by merchant $\mathsf{M}_m \in \mathcal{M}$ in order to join the network. It initially generates a pair of signing/verification keys $(\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm})$ and returns a tuple $(\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm}, \mathsf{pk}_{bm} =\perp)$. The list of keys belonging to the group of merchants is recorded in $\mathcal{M}[\hat{\mathsf{sgk}}_{bi}, \mathsf{vk}_{bi}]_{i=1}^{\ell}$.
- $\mathsf{MRegister}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, \mathcal{M}[\mathsf{vk}_{bi}]_{i=1}^{\ell})$ is executed by any subset of $\mathcal{AU}$ of size at least t to register the merchants who deposit a collateral to join the merchant consortium and assign them a public key pk_{bm}. For each merchant $\mathsf{M}_m \in \mathcal{M}$, it takes the secret signing key of the authorities $\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]$ for $1 \leq i \leq t$, and returns public key pk_{bm} as output. After this phase, the list of parameters for the m^{th} merchant can be updated as $\mathcal{M}[\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm}, \mathsf{pk}_{bm}]$.
- $\mathsf{CuKeyGen}(\mathsf{C}_n)$ is executed by the customers, that for each customer $\mathsf{C}_n \in \mathcal{C}$, a Pseudo-ID generator function (PID) generates an initial secret key $\hat{\mathsf{sk}}_{cn}$ and its corresponding public key pk_{cn}. It returns a tuple of $(\mathsf{pk}_{cn}, \hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn} =\perp)$ with a NIZK proof π_1 to prove that the relation $\mathbf{R}_{\mathbf{L}_1}$ fulfills. The list of customers' keys are kept in $\mathcal{C}[\hat{\mathsf{sk}}_{ci}, \mathsf{pk}_{ci}, \hat{\mathsf{cert}}_{ci} =\perp]_{i=1}^{k}$.

Protocol as Depicted in Fig. 2:

- $\mathsf{CuRegister}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, \mathcal{C}[\mathsf{pk}_{cn}], \pi_1)$ is depicted in step 1 of Fig. 2. It is executed by any subset of $\mathcal{AU}$ of size at least t to certify the public key pk_{cn} of a customer C_n corresponds to some secret value $\hat{\mathsf{sk}}_{cn}$ under the relation $\mathbf{R}_{\mathbf{L}_1}$. Once the customer C_n is registered by the authorities, it receives a certificate $\hat{\mathsf{cert}}_{cn}$ and it updates $\mathcal{C}[\mathsf{pk}_{cn}, \hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}]$.
- $\mathsf{CuCreate}(\mathcal{C}[\hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}])$ is depicted in step 2–3 of Fig. 2. It is executed by the customer to request for certification of their collateral. A successfully registered customer C_n, with certificate $\hat{\mathsf{cert}}_{cn} \neq\perp$, can deposit collaterals in the smart contract. For each deposit j in the smart contract, the customer samples a random value k_j from a uniform distribution $\mathcal{K}_{\mathsf{PRF}}$ in a way that the deposit is not directly linkable to the customer. Then it returns a tuple $\mathcal{CL}[\hat{k}_j, k'_j, \perp]$ as an uncertified collateral along with a proof π_2 depicting the fact that the relation $\mathbf{R}_{\mathbf{L}_2}$ fulfills.

Algorithm 1: Our Construction.

```
Function PGen($1^\lambda, \mathbf{R_L}$):
    $(\vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \hat{\vec{\mathsf{te}}}) \leftarrow \mathcal{ZK}.\mathsf{K}_{\vec{\mathsf{crs}}}(1^\lambda, \mathbf{R_L})$
    $(\mathsf{pp}) \leftarrow \mathcal{TSPS}.\mathsf{Setup}(1^\lambda)$
    return $\mathsf{mpk} := (\mathsf{pp}, \vec{\mathsf{crs}})$
Function AuKeyGen($\mathcal{AU}$):
    $(\hat{\vec{\mathsf{sgk}}}_a, \vec{\mathsf{vk}}_a, \mathsf{vk}_a) \leftarrow$
      $\mathcal{TSPS}.\mathsf{KGen}(\mathsf{mpk}, n, t)$
    return $(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai}, \mathsf{vk}_a]_{i=1}^{n})$
Function MKeyGen($\mathsf{M}_m$):
    $(\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm}) \leftarrow \mathcal{DS}.\mathsf{KGen}(\mathsf{pp})$
    return $(\mathcal{M}[\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm}])$
Function
  MRegister($\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, \mathcal{M}[\mathsf{vk}_{bi}]_{i=1}^{\ell}$):
    for $j \in \mathrm{range}(\ell)$ do
        $(\mathsf{pk}_{bj}, \mathsf{pk}_b) \leftarrow$
          $\mathcal{RRTE}.\mathsf{KGen}(\mathsf{mpk}, \ell, t, 2)$
    return $(\mathcal{M}[\mathsf{pk}_{bi}]_{i=1}^{\ell}, \mathsf{pk}_b)$
Function CuKeyGen($\mathsf{C}_n$):
    $\hat{\mathsf{sk}}_{cn} := \mathsf{PID}(\mathsf{C}_n) \in \mathbb{Z}_{\mathsf{p}}^{*}$
    $\mathsf{sk}'_{cn} \leftarrow \mathcal{CO}.\mathsf{Com}(\mathsf{pp}, \hat{\mathsf{sk}}_{cn})$
    $\mathsf{pk}_{cn} := (\mathsf{sk}'_{cn}, M_1, M_2) \leftarrow$
      $\mathsf{iDH}^{\mathsf{H}}(\mathsf{sk}'_{cn}, \hat{\mathsf{sk}}_{cn})$
    $\mathsf{x}_1 = (\mathsf{pk}_{cn})$
    $\hat{\mathsf{w}}_1 = (\hat{\mathsf{sk}}_{cn})$
    $\pi_1 \leftarrow \mathcal{ZK}.\mathsf{P}(\mathbf{R}_{\mathbf{L}_1}, \vec{\mathsf{crs}}, \mathsf{x}_1, \hat{\mathsf{w}}_1)$
    return $(\mathcal{C}[\mathsf{pk}_{cn}, \hat{\mathsf{sk}}_{cn}], \pi_1)$
Function
  CuRegister($\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, \mathcal{C}[\mathsf{pk}_{cn}], \pi_1$):
    if $\mathcal{ZK}.\mathsf{Vf}(\mathbf{R}_{\mathbf{L}_1}, \vec{\mathsf{crs}}, \mathsf{x}_1, \pi_1) = 1$ then
        $(\hat{\mathsf{cert}}_{cn}) \leftarrow$
          $\mathcal{TSPS}.\mathsf{Sign}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, \mathsf{pk}_{cn})$
    return $(\mathcal{C}[\hat{\mathsf{cert}}_{cn}])$
Function CuCreate($\mathcal{C}[\hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}]$):
    $\hat{k}_j \leftarrow \mathcal{PRF}.\mathsf{KGen}(\mathsf{pp})$
    $k'_j \leftarrow \mathcal{CO}.\mathsf{Com}(\mathsf{pp}, \hat{k}_j)$
    $M_j := (k'_j, M_1, M_2) \leftarrow \mathsf{iDH}^{\mathsf{H}}(k'_j, k_j)$
    $\mathsf{x}_2 = (k'_j)$
    $\hat{\mathsf{w}}_2 = (\hat{k}_j, \hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn})$
    $\pi_2 \leftarrow \mathcal{ZK}.\mathsf{P}(\mathbf{R}_{\mathbf{L}_2}, \vec{\mathsf{crs}}, \mathsf{x}_2, \hat{\mathsf{w}}_2)$
    return $(\mathcal{CL}[\hat{k}_j, M_j], \mathcal{C}[\mathsf{cert}'_{cn}], \pi_2)$
Function AuCreate($\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_a]_{i=1}^{t}$,
  $\mathcal{C}[\mathsf{cert}'_{cn}], \mathcal{CL}[\hat{k}_j, k'_j], \pi_2$):
    if $\mathcal{ZK}.\mathsf{Vf}(\mathbf{R}_{\mathbf{L}_2}, \vec{\mathsf{crs}}, \mathsf{x}_2, \pi_2) = 1$ then
        $(\hat{\mathsf{cert}}_j) \leftarrow$
          $\mathcal{TSPS}.\mathsf{ParSign}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, M_j)$
    return $(\mathcal{CL}[\hat{\mathsf{cert}}_j])$
Function Spend($\mathcal{C}[\hat{\mathsf{cert}}_{cn}, \hat{\mathsf{sk}}_{cn}]$,
  $\mathcal{CL}[\hat{k}_j, \hat{\mathsf{cert}}_j], \mathcal{M}[\mathsf{pk}_{bm}], \mathsf{t}$):
    $(r_{\mathsf{t}}) \leftarrow \mathsf{PRF}_{\hat{k}_j}(\mathsf{t})$
    $R_{\mathsf{t}} := e(r_{\mathsf{t}}, h)$ ▷ $h$ is the generator of $\mathbb{G}_2$.
    $\mathtt{Ct}_m \leftarrow \mathcal{RRTE}.\mathsf{Enc}(\mathsf{pk}_{bm}, \hat{\mathsf{sk}}_{cn}; r_{\mathsf{t}})$
    $\hat{\mathsf{w}}_3 = (\hat{\mathsf{cert}}_j, \hat{k}_j, r_{\mathsf{t}}, \hat{\mathsf{sk}}_{cn})$
    $\mathsf{x}_3 = (R_{\mathsf{t}}, \mathtt{Ct}_m, \mathsf{t})$
    $\pi_3 \leftarrow \mathcal{ZK}.\mathsf{P}(\mathbf{R}_{\mathbf{L}_3}, \vec{\mathsf{crs}}, \mathsf{x}_3, \hat{\mathsf{w}}_3)$
    return $(\mathcal{T}[\pi_3, \mathsf{x}_3, Tx_{ID}])$
Function Vf($\mathcal{M}[\mathsf{pk}_{bm}], \mathcal{T}[\pi_3, \mathsf{x}_3, Tx_{ID}], \mathsf{t}$):
    if $\mathcal{ZK}.\mathsf{Vf}(\mathbf{R}_{\mathbf{L}_3}, \vec{\mathsf{crs}}, \mathsf{x}_3, \pi_3) = 1$ then
        for $i \in \mathsf{StK}$ do
            if $R_{\mathsf{t}} \notin \mathcal{L}_i$ then
                $(\sigma_{R_{\mathsf{t}},i}) \leftarrow \mathcal{DS}.\mathsf{Sign}(\hat{\mathsf{sgk}}_{bi}, R_{\mathsf{t}})$
        if $\mathcal{DS}.\mathsf{Vf}(\mathsf{vk}_{bi}, \sigma_{R_{\mathsf{t}},i}) = 1 \ \wedge \ |\sigma_{R_{\mathsf{t}}}| \geq$
          $(|\mathsf{StK}|/2) + 1$ then
            return 1
Function RevealID($\mathtt{Ct}_m, \mathtt{Ct}_{m'}, v$):
    $(\hat{\mathsf{sk}}_{cn}) \leftarrow \mathcal{RRTE}.\mathsf{Dec}(\mathtt{Ct}_m, \mathtt{Ct}_{m'}, v)$
    return $(\mathsf{sk}_{cn})$
```

- $\mathsf{AuCreate}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, \mathcal{C}[\mathsf{cert}'_{cn}], \mathcal{CL}[\hat{k}_j, k'_j], \pi_2)$ is depicted in step 4–5 of Fig. 2. It is executed by a group of authorities $\mathcal{AU}$ of size at least t. It takes the authorities' secret signing keys $(\hat{\mathsf{sgk}}_{ai})$, an indexed DH message space of PRF key and a NIZK proof π_2 as inputs. To create a certified collateral, it checks the validity of the proof π_2 and whether this collateral exists in the smart contract, and returns certificate $\hat{\mathsf{cert}}_j$ as output. The list of parameters for each collateral is kept by $\mathcal{CL}[\hat{k}_j, k'_j, \hat{\mathsf{cert}}_j]$.
- $\mathsf{Spend}(\mathcal{C}[\hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}], \mathcal{CL}[\hat{k}_j, \hat{\mathsf{cert}}_j], \mathcal{M}[\mathsf{pk}_{bm}], \mathsf{t})$ is depicted in step 6a and 6b of Fig. 2. It is executed by a customer $\mathsf{C}_n \in \mathcal{C}$ who performs a payment to

the merchant $\mathsf{M}_m \in \mathcal{M}$ using the underlying cryptocurrency of their choice at time t. The registered customer uses a certified collateral $\mathcal{CL}[\hat{k_j}, \hat{\mathsf{cert}}_j]$ to provide a payment guarantee to the merchant M_m. The payment made by the customer is always bounded by a publicly known collateral amount. It returns the transaction details as a list of parameters $\mathcal{T}[\mathsf{x}_3, \pi_3, Tx_{ID}]$, which contains a pair of instance and proof (x_3, π_3), along with a set of auxiliary data R_{t}. This function is executed in parallel with an on-chain payment. In particular, the customer must first sign and broadcast an on-chain transaction Tx, and then include its identifier (Tx_{ID}) in the payment guarantee of Spend. The Tx_{ID} could have different formats depending on the underlying blockchain.[1]

- $\mathsf{Vf}(\mathcal{M}[\mathsf{pk}_{bm}], \mathcal{T}[\pi_m, \mathsf{x}_m, Tx_{ID}], \mathsf{t})$ is depicted in step 7–9 of Fig. 2. The merchant $\mathsf{M}_m \in \mathcal{M}$ executes it to check the validity of a received payment guarantee. Once the proof is verified successfully by merchant M_m along with the majority of statekeepers, StK, confirmation that they have not seen a similar payment guarantee in the current epoch (by providing their signatures), the merchant verifies their individual signatures and aggregates them. Once the aggregation is complete, and if Tx_{ID} specifies the merchant's address as the receiver of funds, the merchant provides the items/services to the customer without waiting for the transaction confirmation of the customer's original payment on the blockchain. If the proof verification fails, or the majority of statekeepers do not confirm the guarantee, or the on-chain transaction specifies the wrong receiver address, the merchant rejects the payment guarantee.

Double-Spend Detection as Depicted in Fig. 3:

- $\mathsf{RevealID}(\mathtt{Ct}_m, \mathtt{Ct}_{m'}, v)$ is a deterministic algorithm that takes two ciphertexts $\mathtt{Ct}_m$ and $\mathtt{Ct}_{m'}$ generated under the public key of two distinct merchants M_m and $\mathsf{M}_{m'}$ and returns the plaintext, i.e., the identity of the customer and the secret to their collateral to redeem it. This ID, sk_{cn}, is no longer hidden and can be used by $\mathcal{AU}$ to blacklist the cheating customer and its collateral(s) can be used to remunerate the victim merchant.

3.1 NIZK Languages

In the proposed generic construction in Algorithm 1 we rely on three languages for the NIZK systems, described below.

- Language $\mathbf{L}_1$: Used to prove the correct formation of the customers' public key pk_{cn}, based on the knowledge of secret key $\hat{\mathsf{sk}}_{cn}$. We depict this language formally below, which is used during $\mathsf{CuKeyGen}(\mathsf{C}_n)$.

$$\mathbf{L}_1 = \mathsf{NIZK}\big\{(\hat{\mathsf{sk}}_{cn}) \mid \mathsf{sk}'_{cn} := \mathcal{CO}.\mathsf{Com}(\hat{\mathsf{sk}}_{cn})\big\}$$

[1] For a public blockchain such as Bitcoin or Ethereum, Tx_{ID} could be the hash of a transaction (H[Tx]); for an anonymous blockchain like Zcash, it could be the viewing key of the transaction that enables the merchant to check if he is the receiver of the shielded transaction [19].

- Language $\mathbf{L}_2$: Used to prove eligibility to request a collateral by deriving certificate fulfillment. This language is used during $\mathsf{CuCreate}(\mathcal{C}[\hat{\mathsf{cert}}_{cn}])$.

$$\mathbf{L}_2 = \mathsf{NIZK}\big\{(\hat{k}_j, \hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn})\ |k'_j := \mathcal{CO}.\mathsf{Com}(\hat{k}_j), \hat{k}_j \in \mathcal{K}_{\mathsf{PRF}}, \mathcal{TSPS}.\mathsf{Vf}(\mathsf{pk}_{cn}, \hat{\mathsf{cert}}_{cn}) = 1\big\}$$

- Language $\mathbf{L}_3$: Used to prove the possession of a valid collateral, correctness of PRF evaluation algorithm and RRTE's ciphertext. This language is used during $\mathsf{Spend}(\mathcal{C}[\hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}], \mathcal{CL}[\hat{k}_j, \hat{\mathsf{cert}}_j], \mathcal{M}[\mathsf{pk}_{bm}], \mathsf{t})$.

$$\begin{aligned}\mathbf{L}_3 = \mathsf{NIZK}\big\{(\hat{\mathsf{cert}}_j, \hat{k}_j, r_{\mathsf{t}}, \hat{\mathsf{sk}}_{cn})\mid r_{\mathsf{t}} \leftarrow \mathsf{PRF}_{\hat{k}_j}(\mathsf{t}), R_t := e(r_t, h),\\ \mathsf{Ct}_m := \mathcal{RRTE}.\mathsf{Enc}(\mathsf{pk}_{bm}, \hat{\mathsf{sk}}_{cn}; r_{\mathsf{t}}), \mathcal{TSPS}.\mathsf{Vf}(M_j, \hat{\mathsf{cert}}_j) = 1\big\}\end{aligned}$$

3.2 Security Analysis

Next we formally define the two main security requirements for our construction, namely (1) Anonymity of honest customers and unlinkability of payment guarantees, and (2) Payment certainty for honest merchants. Note that the $\mathsf{AllGen}(.)$ algorithm (see Fig. 4) generates all system setup parameters at once. In the described definitions, it is implicitly assumed that there exists a PPT adversary $\mathcal{A}$ who has access to the following oracles provided by the challenger $\mathcal{B}$:

- **Oracle** $\mathcal{O}_{\mathsf{AuCorrupt}}(Au_i)$: By calling this oracle under the input Au_i, $\mathcal{A}$ can corrupt Au_i and receive its internal states. The set of corrupted authorities is denoted by $\mathcal{AU}'$ and we have $|\mathcal{AU}'| < t$.
- **Oracle** $\mathcal{O}_{\mathsf{CuCorrupt}}(.)$: Adversary $\mathcal{A}$ can corrupt any customer $\mathsf{C}_n \in \mathcal{C}$ by querying this oracle, and receive its uncertified secret key $\hat{\mathsf{sk}}_{cn}$.
- **Oracle** $\mathcal{O}_{\mathsf{ColCorrupt}}(.)$: $\mathcal{A}$ can corrupt at most q_D collaterals $CL_j \in \mathcal{CL}$ to receive their uncertified secret value $\hat{k}_j$. The list of corrupted collaterals is represented by $\mathcal{CL}'$.
- **Oracle** $\mathcal{O}_{\mathsf{MCorrupt}}(.)$: Adversary $\mathcal{A}$ can corrupt a minority set of merchants (statekeepers) like $\mathsf{M}_m \in \mathcal{M}$ and receives its pair of public key pk_{bm} and secret signing key $\hat{\mathsf{sgk}}_{bm}$. The list of corrupted merchants is denoted by $\mathcal{M}'$ s.t. we have, $|\mathcal{M}'| < |\mathsf{stk}|/2$.
- **Oracle** $\mathcal{O}_{\mathsf{Revoke}}(.)$: Adversary $\mathcal{A}$ can revoke at most q_R certified collaterals $CL_j \in \mathcal{CL}$ and redeem the deposited money.
- **Oracle** $\mathcal{O}_{\mathsf{Spend}}(.)$: $\mathcal{A}$ can make at most q_S payment guarantees created by any arbitrary non-corrupted customer to any non-corrupted merchant.

Definition 1 (Payment Unlinkability and Anonymity). *This construction preserves the anonymity of honest customers and provides unlinkability of payment guarantees, if no PPT adversary $\mathcal{A}$ by getting access to $\mathcal{O}_{\mathsf{AuCorrupt}}, \mathcal{O}_{\mathsf{CuCorrupt}}, \mathcal{O}_{\mathsf{MCorrupt}}, \mathcal{O}_{\mathsf{Spend}}$ oracles, $\mathcal{O}_{\mathsf{ANON}}$ in short, and with advantage of $Adv_{\mathcal{A}}^{ANON}(\lambda, \beta) = 2\left((Exp_{\mathcal{A}}^{ANON}(1^\lambda, \beta) = 1) - 1/2\right)$, has a non-negligible chance of winning the experiment described in Fig. 4, i.e. we have, $\left|Adv_{\mathcal{A}}^{ANON}(\lambda, \beta = 0) - Adv_{\mathcal{A}}^{ANON}(\lambda, \beta = 1)\right| \leq \mathsf{negl}(\lambda)$.*

```
$\textsc{Exp}_{\mathcal{A}}^{\text{ANON}}(1^\lambda, \beta)$
$(\mathsf{mpk}, \mathsf{H}, \mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai}, \mathsf{vk}_a]_{i=1}^{n}, \mathcal{M}[\hat{\mathsf{sgk}}_{bi}, \mathsf{vk}_{bi}, \mathsf{pk}_{bi}]_{i=1}^{\ell}) \leftarrow \mathsf{AllGen}(1^\lambda, \mathbf{R_L})$
$(\mathsf{M}_m, \hat{\mathsf{sk}}_0^*, \hat{\mathsf{sk}}_1^*, \hat{k}_0^*, \hat{k}_1^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{ANON}}}(\mathsf{mpk})$
$\beta \leftarrow\$ \{0, 1\}$
$(\pi_\beta, \mathsf{x}_\beta, R_{\mathsf{t},\beta}) \leftarrow \mathsf{Spend}\big(\mathcal{C}[\hat{\mathsf{cert}}_{c\beta}^*, \hat{\mathsf{sk}}_\beta^*], \mathcal{CL}[\hat{k}_\beta^*, \hat{\mathsf{cert}}_\beta^*], \mathcal{M}[\mathsf{pk}_{bm}], \mathsf{t}\big)$
$\beta' \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{ANON}}}\,(\mathcal{T}[\pi_\beta, \mathsf{x}_\beta, R_{\mathsf{t},\beta}])$
$\textbf{return } \beta' == \beta$

$\textsc{Exp}_{\mathcal{A}}^{\text{PC}}(1^\lambda)$
$(\mathsf{mpk}, \mathsf{H}, \mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai}, \mathsf{vk}_a]_{i=1}^{n}, \mathcal{M}[\hat{\mathsf{sgk}}_{bi}, \mathsf{vk}_{bi}, \mathsf{pk}_{bi}]_{i=1}^{\ell}) \leftarrow \mathsf{AllGen}(1^\lambda, \mathbf{R_L})$
$(\pi^*, \mathsf{x}^*, R_{\mathsf{t}}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{PC}}}(\mathsf{mpk})$
$\textbf{if } \mathcal{ZK}.\mathsf{Vf}\,(\mathbf{R_L}, \vec{\mathsf{crs}}, \pi^*, \mathsf{x}^*) == 1 \ \wedge \ q_D < q_S + q_R : \textbf{return } 1$
$\textbf{else return } 0$
```

Fig. 4. Security Games.

Definition 2 (Payment Certainty). *This construction provides payment certainty (PC) if no transaction τ is approved with a non-negligible advantage s.t. $q_S + q_R + \tau > q_D$. No PPT adversary $\mathcal{A}$ with access to $\mathcal{O}_{\mathsf{AuCorrupt}}, \mathcal{O}_{\mathsf{CuCorrupt}}, \mathcal{O}_{\mathsf{ColCorrupt}}, \mathcal{O}_{\mathsf{Revoke}}, \mathcal{O}_{\mathsf{Spend}}$ oracles, $\mathcal{O}_{\mathsf{PC}}$ in short, can win the experiment described in Fig. 4 with a non-negligible advantage in λ and we can write, $Adv_{\mathcal{A}}^{PC}(\lambda) := \Pr[Exp_{\mathcal{A}}^{PC}(1^\lambda) = 1] \le \mathsf{negl}(\lambda)$.*

As a consequence of payment unlinkability and anonymity, no PPT adversary can expose any information about the transaction such as the identity of honest customers or be able to link it to any other transaction made by the customer. As a consequence of payment certainty, no entity, not even after colluding with a group of participants, can transfer and/or revoke more money than the amount deposited. Finally, any system satisfying both these definitions simultaneously reveals the identity of a malicious customer attempting to use one collateral to pay multiple merchants at the same time.

Theorem 1. *The proposed generic construction in Algorithm 1 satisfies the unlinkability and anonymity of payment guarantees as defined in Definition 1.*

Proof. For each payment request, the customer should transfer a tuple $\mathcal{T}[\pi, \mathsf{x}, R_{\mathsf{t}}, H(Tx)]$ where R_{t} is the auxiliary data at time slot t to convince the merchant and the group of statekeepers about the uniqueness of a collateral. Under the existence of a privacy-preserving blockchain Tx_{ID} does not reveal any information beyond the validity of the transaction and it protects the anonymity of the costumers. In this case, to prove that our construction preserves the anonymity of honest customers and provides unlinkability of payments we show that no PPT adversary, $\mathcal{A}$, by providing two pair of challenge secret keys/collateral keys $(\hat{\mathsf{sk}}^*_0, \hat{k}^*_0)$ and $(\hat{\mathsf{sk}}^*_1, \hat{k}^*_1)$, can distinguish between $(\pi_0, \mathsf{x}_0, R_{\mathsf{t},0})$ and $(\pi_1, \mathsf{x}_1, R_{\mathsf{t},1})$ as the output of the spending algorithm. This property is guaranteed because of the following main security properties for the given primitives: Zero-Knowledge property of the given NIZK proof system, computationally hiding property of the given commitment scheme, static-semantically secure property of the given randomness-reusable threshold encryption in bilinear groups and also the weak robustness of the given PRF.

Let the hybrid H^β be the case where the *Anonymity* experiment, $\text{Exp}^{\text{ANON}}_{\mathcal{A}}(\lambda, \beta)$ is run for $\beta = \{0, 1\}$. In this case, we form a sequence of hybrids and show that each of the successive hybrids are computationally indistinguishable from the preceding ones.

- **Hybrid H^β_1:** In this game, we assume the existence of an efficient simulator Sim and then modify the previous hybrid, H^β, by generating the challenge NIZK proof π_β via the simulation algorithm, $\pi'_\beta \leftarrow \mathcal{ZK}.\mathsf{Sim}(\vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \mathsf{x}_\beta)$.

The Zero-Knowledge property of NIZK arguments guarantees that this experiment is indistinguishable from the one for H^β and we can write $H^\beta_1 \approx_\lambda H^\beta$.

- **Hybrid H^2_β:** In this game, we modify H^β_1 s.t. for generating the index *id* the challenger commits $\hat{\mathsf{sk}}^*_{1-\beta}$ instead of $\hat{\mathsf{sk}}^*_\beta$.

According to the hiding property of the given commitment scheme, this experiment is indistinguishable from H^1_β and we can write, $H^\beta_2 \approx_\lambda H^\beta_1$.

- **Hybrid H:** In this game, we modify H^2_β by assuming the challenger runs the RRTE encryption algorithm under the message $m_{1-\beta}$ instead of m_β.

According to the Static Semantic Security property of the proposed randomness-reusable Threshold encryption, this experiment is indistinguishable from H^2_β. To be more concrete, $\mathcal{A}$ cannot distinguish between $\mathtt{Ct}_\beta$ and $\mathtt{Ct}_{1-\beta}$ as long as no twin ciphertext is generated even if the proofs are simulated. Thereby we have, $H_0 \approx_\lambda H^1_0 \approx_\lambda H^2_0 \approx_\lambda H \approx_\lambda H^1_1 \approx_\lambda H^2_1 \approx_\lambda H_1$.

To conclude this security property for the proposed construction, based on the weakly robust property of the given PRF, it is straightforward to demonstrate that the output of a PRF under two distinct keys is computationally indistinguishable and no PPT adversary can distinguish $R_{\mathsf{t},0}$ and $R_{\mathsf{t},1}$. □

Theorem 2. *The proposed generic construction in Algorithm 1 satisfies the payment certainty of payment guarantees as defined in Definition 2.*

Proof. We prove this security property by contradiction and for the simplicity we avoid the hat notion for the secret parameters. Let there is a PPT adversary $\mathcal{A}$ that can break the payment certainty of the scheme and pass the verification phase without meeting at least of the following cases.

- **Case 1.** The adversary $\mathcal{A}$ can forge a valid payment guarantee, $\mathcal{T}$.
- **Case 2.** The adversary $\mathcal{A}$ can forge a valid aggregated signature σ_{R_t} s.t. $|\sigma_{R_\mathsf{t}}| \geq (|\mathsf{StK}|/2) + 1$.

By relying on the existence of a weakly-robust PRF, a *Knowledge Sound* NIZK argument, an existentially unforgeable TSPS construction we show that the success probablity of adversary in "**Case 1**" is negligible. Thus having played a sequence of indistinguishable games between $\mathcal{B}_{\mathrm{WR}}^{\mathrm{PRF}}(1^\lambda), \mathcal{B}_{\mathrm{EUF\text{-}CiMA}}^{\mathrm{TSPS}}(1^\lambda), \mathcal{B}_{\mathrm{KS}}^{\mathrm{NIZK}}(1^\lambda)$ and a PPT adversary $\mathcal{A}$, we gradually turn the payment certainty security game into the security features of the underlying primitives.

- **Game G_0:** In the first security game, let $\mathcal{A}$ forms a challenge transaction τ^* such that $\sum \mathcal{L}_c + \tau^* > \mathsf{col}_\mathcal{A}$ return a valid pair (π^*, x^*) with a non-negligible advantage ϵ. By contradiction, we assume $\mathcal{A}$ can win this game with a non-negligible advantage ϵ and we can write, $Adv_\mathcal{A}^{\mathsf{PC}}(\lambda) = \Pr[\mathcal{A} \text{ Wins } G_0] \geq \epsilon$.
- **Game G_1:** In this game, we modify G_0 such that we assume the existence of an efficient extractor $\mathsf{Ext}(.)$. In this case, there exists an extractor that takes the extraction trapdoor $\hat{\vec{\mathsf{te}}}$ and the received challenge tuple $(\pi^*, \pi_j^*, \mathsf{x}^*)$ as inputs, and returns the corresponding witness $(\mathsf{w}^*) \leftarrow \mathsf{Ext}(\vec{\mathsf{te}}, \mathsf{x}^*, \pi^*)$ s.t. $\mathsf{w}^* = \big(\mathsf{cert}^*, \mu^*, \mathsf{sk}_c^*, r_\mathsf{t}^*, (M_j^*, k^*)\big)$. To be more precise, the extractor first extracts the indexed DH message $M_j^* := (id_j^*, M_{j1}^*, M_{j2}^*)$, and then can extracts the secret PRF key k^* from the proof (π_j^*) as a proof to show the well-formedness of the index id_j^*. The indistinguishability of G_0 and G_1 can be proven via the *Knowledge Extraction* property of NIZK arguments. This property guarantees the existence of the defined extractor under non-falsifiable assumptions and we can write, $Adv_\mathcal{A}^{\mathsf{PC}}(\lambda) = \Pr[\mathcal{A} \text{ Wins } G_0] \approx \Pr[\mathcal{A} \text{ Wins } G_1]$ and this advantage consequently depends on two possible cases,

  $$\Pr[\mathcal{A} \text{ Wins } G_1] = \Pr[\mathcal{A} \text{ Wins } G_1 : (\mathsf{w}^*, \mathsf{x}^*) \in \mathbf{R_L}] + \Pr[\mathcal{A} \text{ Wins } G_1 : (\mathsf{w}^*, \mathsf{x}^*) \notin \mathbf{R_L}].$$

  The probability of an adversary in the latter case can be bounded by the advantage a NIZK's knowledge soundness.

  $$Adv_\mathcal{A}^{\mathsf{PC}}(\lambda) \leq \Pr[\mathcal{A} \text{ Wins } G_1 : (\mathsf{w}^*, \mathsf{x}^*) \in \mathbf{R_L}] + Adv_{\mathcal{B}_{\mathrm{KS}}}^{\mathrm{NIZK}}(\lambda).$$

  Under the assumption that the given NIZK is KS, the adversary $\mathcal{A}$ can win the game when the event of $(\mathsf{w}^*, \mathsf{x}^*) \in \mathbf{R_L}$ occurs.
- **Game G_2:** The challenger for the payment certainty security game can modify G_1 to an attacker against the weakly-robust PRF security game. The intended key k^* is either a valid key $k^* \in \mathcal{K}$ s.t. it is not corrupted by the adversary, i.e. $k^* \notin \mathcal{CL}'$ or it is generated under a random key $k^* \notin \mathcal{K}$. The latter case will be bounded by the advantage of $\mathcal{B}_{\mathrm{WR}}^{\mathrm{PRF}}(1^\lambda)$ attacker, then we can write,

  $$\begin{aligned}&\Pr[\mathcal{A} \text{ Wins } G_2] = \Pr[\mathcal{A} \text{ Wins } G_2 : k^* \in \mathcal{K} \ \wedge \ k^* \notin \mathcal{CL}'] + \\ &\Pr[\mathcal{A} \text{ Wins } G_2 : k^* \notin \mathcal{K}] \leq \Pr[\mathcal{A} \text{ Wins } G_2 : k^* \in \mathcal{K} \ \wedge \ k^* \notin \mathcal{CL}'] + Adv_{\mathcal{B}_{\mathrm{WR}}}^{\mathrm{PRF}}(\lambda).\end{aligned}$$

- **Game G_3:** This is the game G_2, except for a valid pair of witness and statement in $\mathbf{R_L}$ and a fresh and not queried PRF key, one can reduce it to a forgery attack for the underlying TSPS scheme. More specifically, if $k^* \in \mathcal{K}$ and not corrupted before then the challenger can generate its iDH message format M_j^*. Lets the set of authorities indices that are queried before to get a certificate by the adversary is denoted by $\mathcal{S}_{(\star, M_{j2}^*)}$. If $|\mathcal{S}_{(\star, M_{j2}^*)} \cup \mathcal{AU}'| < t$, $\mathcal{B}_{\mathsf{EUF\text{-}CiMA}}^{\mathrm{TSPS}}(1^\lambda)$ returns the pair (M_j^*, cert^*) as a valid forgery for the defined threshold $\mathsf{EUF\text{-}CiMA}$ security game in Def. [16, 4.3]. Thus, we can write,

$$\begin{aligned} &Adv_{\mathcal{A}}^{\mathsf{PC}}(\lambda) \le Adv_{\mathcal{B}_{\mathrm{KS}}}^{\mathrm{NIZK}}(\lambda) + Adv_{\mathcal{B}_{\mathrm{WR}}}^{\mathrm{PRF}}(\lambda) + \Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star, M_{j2}^*)} \cup \mathcal{AU}'| < t] \\ &+ \Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star, M_{j2}^*)} \cup \mathcal{AU}'| \ge t] \le Adv_{\mathcal{B}_{\mathrm{KS}}}^{\mathrm{NIZK}}(\lambda) \\ &+ Adv_{\mathcal{B}_{\mathrm{WR}}}^{\mathrm{PRF}}(\lambda) + Adv_{\mathcal{B}_{\mathsf{EUF\text{-}CiMA}}}^{\mathrm{TSPS}}(\lambda) + \Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star, M_{j2}^*)} \cup \mathcal{AU}'| \ge t]. \end{aligned}$$

Since it is assumed that the adversary $\mathcal{A}$ should provide a fresh and not queried collateral, the probability of the event, "$\mathcal{A}$ Wins $G_3 \wedge |\mathcal{S}_{(\star, M_{j2}^*)} \cup \mathcal{AU}'| \ge t$", is equal to zero. Then we can write,

$$Adv_{\mathcal{A}}^{\mathsf{PC}}(\lambda) \le Adv_{\mathcal{B}_{\mathrm{KS}}}^{\mathrm{NIZK}}(\lambda) + Adv_{\mathcal{B}_{\mathrm{WR}}}^{\mathrm{PRF}}(\lambda) + Adv_{\mathcal{B}_{\mathsf{EUF\text{-}CiMA}}}^{\mathrm{TSPS}}(\lambda).$$

Similarly to demonstrate that the probability of **Case 2** is negligible we rely on the unforgeability of the given aggregatable digital signature. If the adversary $\mathcal{A}$ be able to forge a valid aggregated signature for a majority of the statekeepers then as it is assumed it only can corrupt at most $|\mathcal{M}'| < |\mathsf{StK}|/2$, then we can form an efficient algorithm $\mathcal{B}_{\mathrm{EUF\text{-}CMA}}^{\mathrm{DS}}(1^\lambda)$ to break the EUF-CMA property of the underlying DS scheme. Then we can write:

$$Adv_{\mathcal{A}}^{\mathsf{PC}}(\lambda) \le Adv_{\mathcal{B}_{\mathrm{KS}}}^{\mathrm{NIZK}}(\lambda) + Adv_{\mathcal{B}_{\mathrm{WR}}}^{\mathrm{PRF}}(\lambda) + Adv_{\mathcal{B}_{\mathsf{EUF\text{-}CiMA}}}^{\mathrm{TSPS}}(\lambda) + Adv_{\mathcal{B}_{\mathrm{EUF\text{-}CMA}}}^{\mathrm{DS}}(\lambda).$$

Thus, as long as the underlying primitives are knowledge sound, weakly robust and existentially unforgeable then we can conclude the theorem. □

4 An Efficient Instantiation

In this section, we specify the concrete cryptographic primitives used to instantiate our construction. With the exception of RRTE, which is our novel construction, we refer formal definitions of primitives and their security properties to the full version [28]. We would like to stress the modularity of our construction. The below tools are used in a black box manner and can be replaced by superior tools that future research will inevitably develop.

4.1 Randomness-Reusable Threshold Encryption

(ℓ, t, k)-RRTE is a new observation on threshold encryption (TE) schemes (see the full version [28] for the definition) and enables plaintext confidentiality as long as less than k number of ciphertexts with the same randomness is generated. Once a data owner issues at least k supplementary ciphertexts, it is publicly retrievable and everybody can blind out the encrypted data. We formulate this primitive for compatibility with the rest of our system, but it is worth noting that the underlying idea is similar to offline double spending detection used in e-cash schemes.

Definition 3 (Randomness-Reusable Threshold Encryption). *For a given public parameters* pp *and security parameter* λ*, a* (ℓ, t, k)*-RRTE,* Ψ_{RRTE}*, over the message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$ *consists of three main PPT algorithms defined as follows:*

- $(\vec{\mathsf{pk}}, \mathsf{pk}) \leftarrow \mathcal{RRTE}.\mathsf{KGen}(\mathsf{pp}, \ell, t, k)$*: Key generation is a probabilistic and distributed algorithm that takes* pp *along with three integers* $\ell, t, k \in \mathsf{poly}(\lambda)$ *as inputs. It then returns a vector of public key* $\vec{\mathsf{pk}}$ *of size* ℓ *and a general public key* pk *as outputs.*
- $(\mathtt{Ct}_j, v) \leftarrow \mathcal{RRTE}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m, \mathsf{pk}_j)$*: The encryption algorithm as a probabilistic algorithm takes* pp*, global public key* pk*, a message* $m \in \mathcal{M}$ *along with a public key* pk_j *as inputs. It returns ciphertext* $\mathtt{Ct}_j \in \mathcal{C}$ *associated with the recipient* $j \in \mathcal{R}$ *and an auxiliary value* v *as outputs.*
- $(\perp, m) \leftarrow \mathcal{RRTE}.\mathsf{Dec}(\mathsf{pp}, \{\mathtt{Ct}_j\}_{j \in \mathcal{K}}, v)$*: The decryption algorithm takes* pp*, a set of ciphertexts* $\{\mathtt{Ct}_j\}_{j \in \mathcal{K}}$ *along with an auxiliary value* v *as inputs. If* $|\mathcal{K}| \geq k$*, it returns* $m \in \mathcal{M}$*, else it responds by* $\perp$*.*

Note that in the full version [28] we elaborate more on the security requirements and then we propose an efficient construction based on threshold ElGamal encryptions.

4.2 Pseudo-Random Function

We utilise a weakly-robust PRF proposed by Dodis and Yampolskiy [17] in order to make customers' collaterals reusable. This PRF enables us to define a time of payment, i.e. x in $\mathsf{PRF}_k(x) = g^{1/(k+x)}$ function and prove the validity of operations, efficiently. This ensures that a customer always has to input the time of payment, which is then verified by a receiving merchant. By utilizing this property and its combination with RRTE, as discussed in Sect. 1, we can block a customer from reusing the same collateral for a pre-defined time period.

4.3 Digital Signature Schemes

We require two types of signatures, one for the authorities and another for the statekeepers. These signatures need non-overlapping properties which we detail below.

- Threshold Structure-Preserving Signatures [16]. There are two reasons to use the TSPS scheme proposed by Crites et al. Firstly, like any other digital signature, it provides authentication, such that no entity except the qualified authorities can issue collateral proofs. Secondly, due to its threshold nature, TSPS enables our construction to rely on an honest majority (authorities) instead of a central trusted party.
- BLS Signatures [8]. BLS Signatures are efficiently aggregatable, and thus they are useful in our setting. A statekeeper must validate payment requests from various merchants, and they do so using BLS signatures. For a victim merchant to redeem user collateral from the smart contract, they may first aggregate the signatures allowing for a shorter interaction.

4.4 Commitment Scheme

In our construction, we use the Pedersen commitment scheme [32] due to the following reasons; firstly, the TSPS construction is defined over the indexed DH message spaces (see the full version for the exact definition) and each secret PRF key needs to get an index. Hence, these commitments are used to the secret scalar PRF keys as an index. Secondly, the hiding property of such commitments masks the secret PRF keys used in our construction. In addition, the binding property of Pedersen commitments ensures the unforgeability of these secret PRF keys. Finally, Pedersen commitments are compatible with discrete logarithm-based proofs like original Sigma protocols and enables customers to efficiently prove knowledge of these committed values.

4.5 NIZK Proofs

To instantiate the described NP-relations in Sect. 3.1, we utilize three main proof systems: Sigma protocols [34], range-proofs [10] and GS proof systems [24] (see the full version). Sigma protocols are an efficient choice as the main proof system in our implementation; we use the Fiat-Shamir heuristic [22] to make then non-interactive. Range-proofs enable us to prove that a hidden value lies in a range interval. GS proof systems are useful as they are secure in the standard model and support a straight-line extraction of the witnesses. Additionally, the instantiation of these proofs does not require any trusted setup and can be batched: this enables an efficient verification [26].

5 Performance Analysis

In this section, we demonstrate the performance of our system. Based on the application, the costs incurred in each phase are divided into two parts, termed "offline phase" and "online phase". The former includes the parameter generation, key generation and registration functions. The latter is solely responsible for spending and verification and is the main focus of this evaluation.

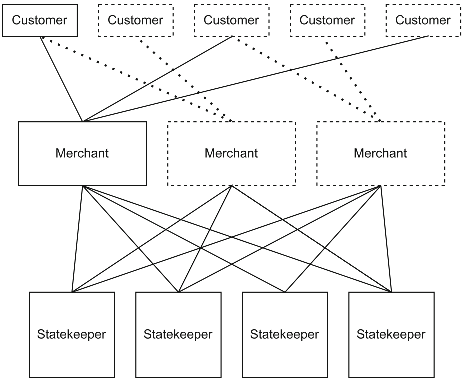


Fig. 5. Experimental setup. Dotted lines and shapes represent parties and workload that do not affect the leftmost customer or the leftmost merchant. Our experiment only considers the solid lines and shapes. From the perspective of the leftmost client and the leftmost merchant, this is equivalent to running the full system. While statekeepers are the same as merchants, we make these two sets distinct for clarity.

Our experimental setup is similar to the one in Snappy. Namely, we distribute various parties in different regions around the world and measure the end to end latency of transactions[2]. Specifically, our implementation uses the Charm-Crypto framework [15], a Python library for Pairing-based Cryptography and obtained the benchmarks on four AWS EC2 instances. The scenario we consider is similar to the one in Snappy. Namely, merchants, customers and statekeepers are globally distributed in four different locations and we create 1 000 tps in order to measure the average time it takes for one transaction to complete. Since transactions are distributed to many merchants and the merchants run independently, it is possible to create an equivalent scenario and only consider the work needed for one merchant. Consider the workload from the perspective of a single statekeeper: its workload depends on transactions that are passing through all other merchants. To accurately estimate the workload of a single statekeeper, we injecting artificial verification requests to it. Our scenario is summarized in Fig. 5.

All our EC2 instances had the same computational configuration, i.e., an Ubuntu Server 20.04 LTS (HVM) with an Intel (R) Xeon(R) CPU @ 2.50 GHz and 16 GB of memory. We apply the Barreto-Naehrig (BN254) curve (also known as type F groups), $y^2 = x^3 + b$ with embedding curve degree 12 [3]. In this pairing group, the base field order is 256 bits.

[2] The open-source implementation can be found in this repository.

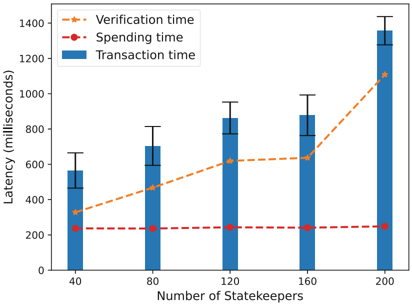


Fig. 6. Latency comparison. Total transaction time for 1000 tx per second vs the number of statekeepers.

Latency. As illustrated in Fig. 6, the latency for each transaction grows linearly with the number of statekeepers verifying this transaction (depicted with the orange line). During our evaluations, we noticed that the time required for a customer to generate and send a payment guarantee (depicted with the red line) is mostly constant, i.e., ∼240 ms. However, in the case of 40 statekeepers, our construction allows a customer and merchant to successfully transact within ∼550 milliseconds (ms). In contrast, in the case of 200 statekeepers, a transaction takes ∼1.3 seconds (s). Hence, the time required to guarantee a payment to merchants in our construction is bounded by the set of statekeepers. A direct comparison with Snappy is not possible for two reasons; firstly, the evaluation of Snappy only considers the time taken for payment approval, i.e., it does not consider the time spent on customer-merchant interaction. Secondly, their simulation code is not freely available. A standalone analysis shows that our construction provides privacy against statekeepers without impacting the latency of payments. Greater number of statekeepers provides more robustness and better protection against double-spends, but requires stronger liveness assumptions on top of increasing the latency. We find 120 statekeepers to be an optimal trade-off of these factors, ultimately leading to latency lower than 1 s.

Smart Contract Cost. The transition between states happens depending on the function calls on the smart contract. For simplicity, we describe here only the functionality of the smart contract focusing on one customer and multiple merchants. We refer to our smart contract as $Auth_{SC}$, an entity is referred to as e_x where $x = c$ or m for customer or merchant respectively. The underlying ledger is referred to as L_{SC} and an entity's account on that ledger is referred to as Acc_L^x where $x = c$ or m respectively. The private ledger of the merchants is referred to as $Bull_m$, since it behaves like a bulletin board. $Auth_{SC}$ has seven states as follows:

init: $Auth_{SC}$ is deployed. e_c can now deposit funds (col_c). If so, then change state to *ready*. Else do not change state.

ready: e_c successfully registers by depositing col_c in $Auth_{SC}$. If col_c is available in $Auth_{SC}$, change state to *pay*. Else do not change state.

pay: If e_c has made payment (pay_{m_i}), change state to $reclaim_m$. Else do not change state.
reclaim$_m$: Check $Bull_m$ for double-spends from e_c. If double-spend present, use secret to reclaim pay_{m_i} and change state to *withdraw*. If no double-spend found until actual payment received, change state to $reclaim_c$.
reclaim$_c$: If 1 day has passed since pay_{m_i}, reclaim pay_{m_i} and add it to col_c. Then, change state to *withdraw*. Else do not change state.
withdraw: If e_c wants to exit the system and the state is *withdraw* or *ready*, send money from $Auth_{SC}$ to Acc_L^c and change state to *exit*. If e_w wants to exit the system and the state is *withdraw*, send money from $Auth_{SC}$ to Acc_L^w and change state to *exit*. Else do not change state.
exit: Remove e_x from $Auth_{SC}$ and change state to *init*.

Table 1 lists the gas fees of executing various functions of our system. In the first column, we mention the amount of base gas fee, and then we express it as a proportion of minimum gas fee of a transaction. The minimum gas fee is set to 21 000 GWei, which is also the amount of gas that standard on-chain payments require. Note that the table reflects the fees users can expect to pay, although the actual amount also depends on the so-called priority fee which depends on the current traffic in the Ethereum transaction market. More information about the costs incurred due to our SC is given in Sect. 6.

Table 1. Costs of transactions on our smart contract deployed on Ethereum, and the cost as a proportion of a standard transaction $\Delta = 21\,000$.

Function	Customer reg.	Pay	Merchant reg.	Reclaim	Withdraw Bal.
Gas	107 400	44 055	54 317	34 972	22 352
$\times\Delta$	5.1	2.09	2.59	1.65	1.06

6 Discussion

6.1 Collateral Reusability

This property can be understood by comparing to PCNs such as Lightning [33], Raiden [30]. Our work is similar to PCNs since we are both reliant on collaterals for guaranteeing security of transactions. PCNs enable quick and cheap transactions over established channels between parties (routes), but suffer from route availability issues in case any involved party is unresponsive. They are capital dependent, since each channel in a path must individually have sufficient capital to route a certain transaction [25].

The main point of difference between our work and PCNs is the suitability for supporting retail markets. The capital locked into a payment channel is only suitable for a specific kind of payments, while our collaterals allow customers to pay any merchant in the system. This is because PCNs aren't designed to tackle the unilateral nature of payments in retail markets. The funds locked in a channel deplete quickly, and hence their capability to act as intermediaries decreases [29]. Although fund rebalancing techniques [4] exist to mitigate channel depletion, they require a user to have multiple channels and are ineffective for managing unilateral payments. Rebalancing is generally not possible for a user that only employs their channels for payments and not for getting paid. Our protocol is tailor made to this economic environment, and so we believe it supplements PCNs, rather than directly competing with them.

6.2 Trust Assumptions

There is a general trade-off between the efficiency and privacy of a financial system and the level of trust assumed between participants. For instance, a trusted central entity can efficiently set up a digital currency system, as evidenced by Chaumian e-cash. Measures of transaction latency and throughput thrive at the high cost of trust in the central authority. On the other end, decentralized blockchains achieve functional but slow financial systems without requiring trust in any single party.

The performance of our construction is based on a set of trust assumptions. Our architecture is semi-decentralized in the sense that we rely on an honest majority of authorities to initialize our construction. This is similar to the approach of LDSP [31] with the crucial distinction that our authorities do not play any role in our payment protocol. The merchants and statekeepers have greater power to punish dishonest customers by confiscating their collateral. Yet, we allow an honest majority of merchants to do so *only* against customers who attempt to double-spend, not honest customers. Moreover, the design is permissionless in that cryptocurrency holders can freely participate as customers. Considering the general trade-off between centrality, trust, performance and efficiency, we consider our setup to lie in a "sweet spot" where balance is achieved through cryptographic innovations. We call for further cryptographic innovations and welcome research into even more trustless, robust and secure systems.

A similar trade-off has been observed in PCN, a promising scalability solution for cryptocurrencies, by Avarikioti et al. [2], who suggest that PCN are more stable and efficient when centralized structures are present. In an empirical survey, Zabka et al. [35] observe the rising centrality in the Lightning Network as the capacity and capabilities of Lightning grew over time.

6.3 Privacy

The main idea underlying our private double-spending protection, goes all the way back to the e-cash schemes introduced by Chaum [14]. The approach of realizing offline payments while detecting double-spending, known as the Chaum-Fiat-Naor (CFN) approach was adopted and improved by several following e-cash systems [5,9,12,13,23]. The existing plethora of literature has made several improvements to Chaum's e-cash, however, all work with centrally issued currency and mostly rely on a custodian bank to catch double-spending. Our work is also an application of the CFN approach, with the major difference of building upon decentralized cryptocurrencies for safely reducing latency, instead of building an entire e-cash scheme from the ground up. When compared to double-spend detection techniques in e-cash (for instance the one recently used in [6]), our novel RRTE is more efficient in terms of communication rounds, allows the deposited collaterals to be reused and by default enables anyone to track double-spends.

However, *on-chain privacy* is derived from the underlying blockchain, and is the highest level of privacy that one can hope to achieve at the protocol level. In other words, implementing our overlay on a completely de-anonymized and public blockchain cannot make the payments private, since the underlying blockchain will reveal private data no matter how secure the protocol. Similarly, developing on top of private blockchains such as Monero doesn't directly solve the privacy issues of earlier works that allowed transactions of the same user to be linked. In this way, the question of blockchain-level privacy is relevant yet orthogonal to our work. While Snappy claims that future improvements such as deployment on privacy-preserving blockchains that support privacy-preserving SC will enable their construction to provide on-chain privacy, that is not true. A detailed explanation is given in Sect. 2 of the full version [28].

6.4 On-Chain Transaction Fees

The transaction fees in our system differ from conventional fees since the customer pays the merchant indirectly through a smart contract (SC). This is necessary to prevent an on-chain double-spend by a malicious customer. By on-chain double-spend, we mean to distinguish between a double-spend attempt of customer collateral, and a double-spend on the underlying blockchain itself. Even if a malicious customer can influence miners and induce a blockchain double-spend, the SC-based transaction is able to remunerate the affected merchant. Simply put, the SC can escrow the funds until sufficient confirmations of on-chain payment have been found. We stick to the convention of 6 succeeding blocks after said transaction. As discussed in Sect. 5, executing payment through our SC incurs twice the on-chain transaction fees of a standard on-chain Ethereum payment. There is an additional fee incurred by the merchant during withdrawal of payments, but this is far less frequent than the former. Nevertheless, it is desirable to construct a more cost efficient yet secure system for direct customer to merchant payments.

A potential fix could be to encode specific spend conditions for user collaterals. To be precise, any merchant can move the collateral by providing evidence of a conflicting transaction on the blockchain. This could be implemented via a payment guarantee to said merchant, along with on-chain evidence of a conflicting payment. We leave this implementation, along with other possible optimizations of fees, for future work.

6.5 Incentives of Involved Parties

This work deals with the cryptographic challenges of achieving privacy while reducing latency of cryptocurrency payments. Our focus is admittedly myopic, as we overlook practical aspects of incentives. For instance, we refer to authorities that register merchants and customers, but these authorities lack a concrete incentive to fulfil this role honestly. As an initial and arbitrary choice, we selected a subset of involved merchants to play this role of authorities, while requiring an honest majority of authorities. It is unclear who should be playing this role, and what their incentives should be. Could we perhaps allocate a small fee per merchant to authority? Or automatically grant a fraction of collaterals confiscated from dishonest users? Or even eliminate this issue entirely by building more advanced cryptography so that our overlay can be set up even without their existence?

Similarly, we lack a clear explanation of incentives for statekeepers. A basic solution would be to allocate a certain fraction of each transaction value to the statekeepers; however, this still needs to be properly analyzed in order to confirm if such an incentive is sufficient.

7 Conclusion

In this paper, we present a new overlay network for instant confirmation of cryptocurrency transactions, that also maintains anonymity of users and unlinkability of their transactions. On the one hand, it allows merchants in a retail system to safely accept fast payments without risk of double-spending. On the other, dishonest customers who attempt to double-spend get their identities exposed and their collateral confiscated to reimburse the merchants. Honest customers, however, are able to *reuse* their collaterals.

To this end, we designed a novel randomness-reusable threshold scheme, that enables participants to audit the payments in the network and reveal the identity of malicious customer who perform double-spending. This threshold encryption scheme maintains the privacy of honest customers who do not attempt to double-spend. We provide a formal proof of security with respect to three main features namely *customers' anonymity, unlinkability of transactions* and *payment certainty for merchants.* We motivate our choice of cryptographic primitives and efficiently implement them. Our evaluation shows that our construction allows for fast global payments with a delay of less than 1 s.

Acknowledgment. We would like to thank Svetla Nikova, Philipp Jovanovic, Christian Badertscher and Daniel Slamanig for the helpful discussions and the anonymous reviewers for their valuable comments. Akash Madhusudan, Mahdi Sedaghat and Bart Preneel were supported in part by the Flemish Government through the FWO SBO project SNIPPET S007619, the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058 and by CyberSecurity Research Flanders with reference number VR20192203. Samarth Tiwari was supported by ERC Starting Grant QIP–805241.

Kelong Cong was supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. FA8750-19-C-0502. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA, the US Government, Cyber Security Research Flanders or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

1. Aumayr, L., Abbaszadeh, K., Maffei, M.: Thora: Atomic and privacy-preserving multi-channel updates. IACR Cryptol. ePrint Arch. 317 (2022). https://eprint.iacr.org/2022/317
2. Avarikioti, Z., Heimbach, L., Wang, Y., Wattenhofer, R.: ride the lightning: the game theory of payment channels. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 264–283. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_15
3. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006). https://doi.org/10.1007/11693383_22
4. Avarikioti, Z., Pietrzak, K., Salem, I., Schmid, S., Tiwari, S., Yeo, M.: HIDE & SEEK: privacy-preserving rebalancing on payment channel networks. Cryptology ePrint Archive, Report 2021/1401 (2021). https://eprint.iacr.org/2021/1401
5. Baldimtsi, F., Chase, M., Fuchsbauer, G., Kohlweiss, M.: Anonymous transferable E-cash. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 101–124. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_5
6. Bauer, B., Fuchsbauer, G., Qian, C.: Transferable E-cash: a cleaner model and the first practical instantiation. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12711, pp. 559–590. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_20
7. Benmeleh, Y.: Blockchain firm starkware valued at $2 billion in funding round (2021). https://www.bloomberg.com
8. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. J. Cryptol. **17**(4), 297–319 (2004). https://doi.org/10.1007/s00145-004-0314-9
9. Brands, S.: Untraceable off-line cash in wallet with observers. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_26
10. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press (2018). https://doi.org/10.1109/SP.2018.00020
11. Buterin, V.: An incomplete guide to rollups (2021). https://vitalik.ca/general/2021/01/05/rollup.html
12. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact E-cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_18
13. Canard, S., Gouget, A.: Multiple denominations in E-cash with compact transaction data. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 82–97. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_9

14. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO 1982, pp. 199–203. Plenum Press, New York (1982)
15. Joseph, A.A., et al.: Charm: a framework for rapidly prototyping cryptosystems. J. Cryptograph. Eng. **3**, 111–12 (2013). https://doi.org/10.1007/s13389-013-0057-3
16. Crites, E., Kohlweiss, M., Preneel, B., Sedaghat, M., Slamanig, D.: Threshold structure-preserving signatures. Cryptology ePrint Archive, Paper 2022/839 (2022). https://eprint.iacr.org/2022/839
17. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30580-4_28
18. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: virtual payment hubs over cryptocurrencies. In: 2019 IEEE Symposium on Security and Privacy, pp. 106–123. IEEE Computer Society Press (2019). https://doi.org/10.1109/SP.2019.00020
19. Electric Coin Company: Explaining viewing keys. https://electriccoin.co/blog/explaining-viewing-keys/. Accessed 13 Feb 2023
20. Ethereum.org: Layer 2 rollups (2021). https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/
21. ethos.dev: The beacon chain ethereum 2.0 explainer you need to read first. https://ethos.dev/beacon-chain. Accessed 13 Feb 2023
22. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
23. Frankel, Y., Tsiounis, Y., Yung, M.: "Indirect discourse proofs": achieving efficient fair off-line e-cash. In: Kim, K., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 286–300. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0034855
24. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_24
25. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: layer-two blockchain protocols. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 201–226. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_12
26. Herold, G., Hoffmann, M., Klooß, M., Ràfols, C., Rupp, A.: New techniques for structural batch verification in bilinear groups with applications to Groth-Sahai proofs. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 1547–1564. ACM Press (2017). https://doi.org/10.1145/3133956.3134068
27. Hill, K.: How target figured out a teen girl was pregnant before her father did. https://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/?sh=53d927356668. Accessed 30 Aug 2022
28. Madhusudan, A., Sedaghat, M., Tiwari, S., Cong, K., Preneel, B.: Reusable, instant and private payment guarantees for cryptocurrencies. Cryptology ePrint Archive, Paper 2023/583 (2023). https://eprint.iacr.org/2023/583
29. Mavroudis, V., Wüst, K., Dhar, A., Kostiainen, K., Capkun, S.: Snappy: fast on-chain payments with practical collaterals. In: NDSS 2020. The Internet Society (2020)
30. Network, Raiden: What is the raiden network (2019). https://raiden.network/101.html

31. Ng, L.K.L., Chow, S.S.M., Wong, D.P.H., Woo, A.P.Y.: LDSP: shopping with cryptocurrency privately and quickly under leadership. In: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), pp. 261–271 (2021). https://doi.org/10.1109/ICDCS51616.2021.00033
32. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
33. Poon, J., Dryja, T.: The Bitcoin lightning network: scalable off-chain instant payments (2016). https://lightning.network/lightning-network-paper.pdf
34. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
35. Zabka, P., Foerster, K.T., Schmid, S., Decker, C.: A centrality analysis of the lightning network (2022). https://doi.org/10.48550/ARXIV.2201.07746, https://arxiv.org/abs/2201.07746

System Security

BinAlign: Alignment Padding Based Compiler Provenance Recovery

Maliha Ismail[1](✉), Yan Lin[2], DongGyun Han[3], and Debin Gao[1]

[1] Singapore Management University, Singapore, Singapore
{malihai.2020,dbgao}@smu.edu.sg
[2] College of Cyber Security, Jinan University, Guangzhou, China
yanlin@jnu.edu.cn
[3] Royal Holloway, University of London, London, UK
DongGyun.Han@rhul.ac.uk

Abstract. Compiler provenance is significant in investigating the source-level indicators of binary code, like development-environment, source compiler, and optimization settings. Not only does compiler provenance analysis have important security applications in malware and vulnerability analysis, but it is also very challenging to extract useful artifacts from binary when high-level language constructs are missing. Previous works applied machine-learning techniques to predict the source compiler of binaries. However, most of the work is done on the binaries compiled on Linux operating system. We highlight the importance and need to explore Windows compilers and the complicated binaries compiled on the latest versions of these compilers. Therefore, we construct a large dataset of real-world binaries compiled with four major compilers on Windows and four most common optimization settings. The complexity of the optimized programs leads us to identify specific patterns in the binaries that contribute to source compiler and specific optimization level. To address these observations, we propose an improved model based upon the state-of-the-art, and incorporate streamlined alignment padding features in the existing model. Thus, our improved model learns alignment instructions from binary code of portable executables and libraries using the attention mechanism. We conduct an extensive experimentation on a dataset of 296,169 unique and complex binary code generated from C/C++ applications. Our findings demonstrate that our proposed model significantly outperforms the state-of-the-art in accurately predicting the source compiler and optimization flag for complex compiled code.

Keywords: compiler provenance · alignment padding · Windows binaries · binary code similarity

1 Introduction

Microsoft Windows, which is currently the most dominant desktop operating system, commands a substantial market share.[1] However, it is also one of the most

[1] https://gs.statcounter.com/os-market-share/desktop/worldwide (Windows).

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 609–629, 2023.
https://doi.org/10.1007/978-3-031-35486-1_26

targeted platforms for malicious activities by hackers and attackers. Statista[2] reports that 91% of newly developed ransomware in 2022 is intended to target Windows operating system. As a result, security analysts and researchers are interested to unleash all those methods that can aid in identifying the characteristics of binary code available in the wild.

When compiling a C/C++ source application, several flags are passed to the compiler, signaling the developer's intention to either keep or drop some information or to modify the original code in a more optimized version. The executable binary does not need to have knowledge of the compilation flags once it is compiled, as this information is no more required to execute the binary. However, these flags are useful during analysis to investigate whether a file was compiled with a specific flag that could expose vulnerabilities [7,32]. They are also useful, when compiler-optimization-specific security policies are applied on applications at the binary level for efficient CFI enforcement [17,18]. Thus, the security policies are applied varyingly for different compiler-optimization settings. Compiler provenance answers fundamental questions of malware analysis and software forensics, to know whether binaries are generated by similar toolchains [5,6,12,16,25,28–31]. It also aids the development of binary tools and analysis capabilities targeted at specific compilers or source languages [7,28,32]. Furthermore, the source compiler, version and development environment of binaries are amongst the most fundamental artifacts required for analysis at binary level.

Several studies have been conducted to develop techniques for compiler provenance analysis, which aim to identify source compilers and optimization flags [11,12,14,20,21,26,34,40]. However, the accuracy of these techniques may reduce due to the continuous developments in modern C/C++ optimizing compilers. Thus, with the advent of latest optimization strategies and newer Intel architectural extensions, the compiler provenance analysis approaches may become outdated [10,23]. While previous research into compiler provenance analysis techniques [5,25,28,30,31,33] demonstrate high accuracy and promising results, the study of compiler provenance on Windows platform is limited. Thus, we analyze the binaries compiled with Windows compilers and found significant patterns that could be a good indicator of the compiler and optimization levels.

Alignment padding [10,23] is a prominent feature of Windows binaries generated by modern compilers as it aligns the addresses for faster memory access and prevents processor faults. Since compilers optimize the code for either speed or size, their choice of instructions and preference for keeping data in the data section or in-line within the code determines the variation of alignment padding among compilers at different optimization settings. What makes the alignment padding interesting is the form and number of bytes emitted by the compiler with respect to the optimization [10,23]. Therefore, we identified four distinguishing patterns of alignment padding and integrate them into state-of-the-art model for learning and classifying the source compiler and optimization level.

[2] https://www.statista.com/statistics/701020/major-operating-systems-targeted-by-ransomware/.

With this observation, we propose BinAlign that leverages alignment padding in the binary code to help predict source compiler and optimization flags of the compiled code with improved accuracy. Thus, we train our proposed model to map the alignment padding in order to classify source compiler and optimization flag of the target binaries. The classification task identifies the toolchain used to generate unknown binaries and produces information that is required by binary analysis tools (i.e., for CFI enforcement and security patching) [17]. Thus, in order to evaluate the performance of o-glassesX [25] and our improved model, we mainly focus on answering the following research questions:

- RQ1. How effective is BinAlign in identifying the source compiler of a binary?
- RQ2. How effective is BinAlign in identifying two- (i.e., Od/O0 and Ox/O3) and four-level[3] (i.e., Od/O0, O1, O2 and Ox/O3) of compiler optimization settings?
- RQ3. How effective is BinAlign in identifying the joint source compiler and optimization settings?

Hence, we base our study on a set of 296,169 real-world binaries compiled with four major compilers, i.e., Microsoft Visual C++ (MSVC) [1], Clang-cl (Clangcl)[4], Intel C Compiler (ICC)[5], and Minimalist GNU Compiler Collection for Windows (MinGW) [2], in two- and four-level of varying compiler optimization settings. We achieved an overall f-measure (F1) of 0.978 when predicting the source compiler among four compilers and two compiler optimization levels. The main contributions of our paper are as follows:

- **An in-depth study on compiler provenance of binary code:** We investigate the characteristics of Windows binaries with the perspective of alignment bytes generated by modern compilers at four levels of compiler optimization settings.
- **Alignment padding based compiler provenance model:** We propose an improved convolutional neural network (CNN) that learns the special characteristics of alignment padding in the binaries and results in an improved performance.
- **Generality of our approach:** In order to evaluate the applicability of our method, we gather a collection of benign and malicious software binaries from real-world applications, compiled using different versions of the MSVC compiler. This helps us enhance the precision of our approach when analyzing complex and obfuscated malware binaries.

[3] For MSVC, Clangcl and ICC compilers, the optimization levels Od and Ox refer to *none* and *extreme* level of optimization. However, for the MinGW compiler, the corresponding optimization levels are O0 and O3, respectively.

[4] https://clang.llvm.org/docs/MSVCCompatibility.html (Clangcl with MSVC).

[5] https://support.alfasoft.com/hc/en-us/articles/360002874938 (Intel compatibility with MSVC).

Table 1. Alignment padding frequently found in common sections of the PE binary

Category	Section	Content	Paddings
Code section	`.text`	Executable code	`INT3`, `NOP`
Data Sections	`.data`	Read-write initialized data	`ZERO, DB`
	`.pdata`	Exception information	`ZERO, DB`
	`.rdata`	Read-only initialized data	`ZERO, DB`
	`.idata`	Import tables	none

2 Background and Related Work

2.1 Alignment Padding

In this section, we revisit the alignment padding generated by compilers in the binaries for aligning code and data. The term *Alignment padding* is referred to as the padding code placed as trailing sequence next to a control-flow transfer, or padding bytes at specific locations to align data and instructions in the binary (See Footnote 5). Alignment padding consists of an opcode and optionally an operand, similar to binary instructions on any architecture or platform [3].

Alignment Padding in PE Sections. Portable executable (PE) is a file format for executables, object code, and dynamic link libraries (DLLs) on Windows. Table 1 shows the sections commonly found in a PE binary and the types of alignment padding[6] that are most frequently found in each section. As described in Microsoft developer documentation [1], each section consists of different types of program data. The `.text` section contains executable code, while the data sections maintain data to execute the binary (i.e., `.data`, `.pdata`, `.rdata`, and `.idata`). Alignment padding is found in both the `.text` and data sections except for the `.idata` section, which contains the import directory and import address table (IAT). Table 2 shows the placement of four major types of alignment padding and filler instructions in the binary.

2.2 Machine Learning Approaches for Compiler Provenance

This section revisits the previous research conducted on compiler provenance and introduces the relevance of alignment padding for provenance recovery. Previous works utilized machine learning (ML) methods to perform compiler provenance recovery, since signature-based[7,8] methods depend on signatures database [9] to report the source compiler. On the other hand, ML-based methods learn the compiler-specific patterns and features to predict the source compiler of previously unseen binaries. Rosenblum et al. [30] were the first to extract syntactic and

[6] We use the instructions to represent alignment padding except `ZERO` as the type of alignment padding.

[7] https://github.com/horsicq/Detect-It-Easy.

[8] https://www.aldeid.com/wiki/PEiD.

Table 2. Types of alignment padding in the binary code

#	Align	Purpose	Usage
1	`NOP`	Program execution continues with the next instruction	Function-entry alignment in ICC and MinGW
2	`INT3`	Single-byte instruction for setting breakpoints for the debugger	Function-entry alignment in MSVC, Clangcl, ICC
3	`DB`	Reserve space for data	Data alignment in `.text` and data sections for MSVC, Clangcl, ICC, whereas data alignment in data sections only for MinGW
4	`ZERO`	`ADD` instruction with zero opcode and zero operand	Align code and data; may also update memory location, set carry, overflow, and zero flags
5	Filler	Pseudo NOPs	`MOV RAX,RAX`; `LEA RBX,[RBX+0]`; `XCHG RAX,RAX`

structural features from the binaries based on instruction idioms and graphlets. This work was followed by Chaki et al. [6], Xue et al. [39], and Rahimian et al. [27] that used various machine learning classifiers to identify similar chunks of code in the binary. Moreover, the past works [20,21,26,30,31,33,34] used binary-level control flow features and functions to predict program provenance. More recent works such as Ding et al. [8] proposed an Asm2Vec model for assembly code learning based on functions.

Although, past research acknowledged the alignment padding patterns in the binary [4,31,36,37], the significance of these patterns in relevance to program provenance has not been considered earlier. Rosenblum et al. [31] named the alignment bytes as gaps within the functions, whereas Andriesse et al. [4] considered these patterns as inline data within the code. Wang et al. [36,37] corroborated the reassembly of binary, while considering the memory alignment of data and function pointers. While considering all the past efforts, the compiler provenance recovery models [20,21,26,30,33,34] that take the control-flow features of the binary ignore the alignment padding that lie outside the control flow graph of the program such as function-entry and loop-entry alignment.

In this work, we chose o-glassesX [25] as our evaluation baseline for compiler provenance recovery leveraging alignment padding. This is because o-glassesX is state-of-the-art model with 97% accuracy, while utilizing short binary code from C/C++ object files. However, on recent compiler versions and a complex set of binaries, o-glassesX does not maintain to achieve the claimed accuracy. Therefore, we propose BinAlign to predict the source compiler and optimization level with better prediction accuracy.

2.3 o-glassesX Architecture

This section briefly explains the architecture of our baseline, o-glassesX. It uses natural language processing (NLP) techniques called the attention mechanism with convolutional neural network (CNN) to capture the characteristics of a single instruction by multiple local receptive fields [25]. The input unit of CNN is the local receptive field (i.e., kernel), whereas the output unit is the volume of kernel size (`K`), depth, and stride (`S`) length. The depth of the CNN refers to the number of filters, whereas stride is the step size of the kernel when traversing the width and height of the input binary image. In the preprocessing stage, o-glassesX disassembles binary into 128-bit fixed length instructions padded with zeros to construct a 2048-bit sequence of 16 instructions. The underlying architecture of neural network comprises the following CNN layers:

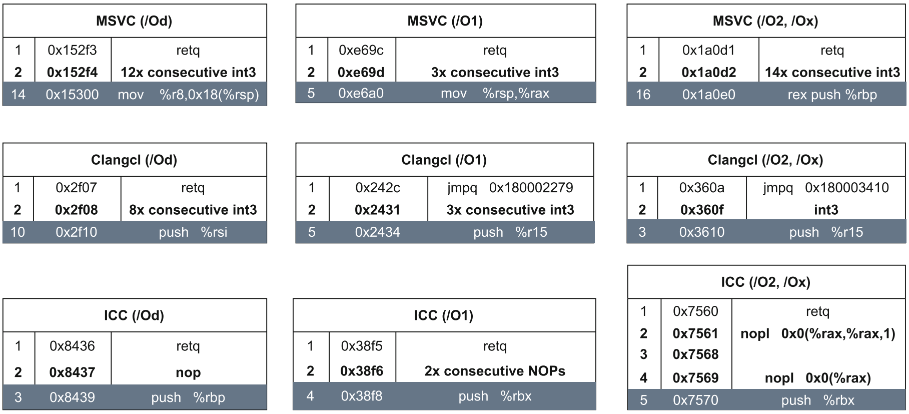
- The first layer takes 2048 bits of binary code as input. Each unit is a single-dimension 128-unit kernel, with stride length of 128 and depth of 96.
- The second layer is the positional encoding layer with 256 filters to a 16 × 96 input volume with a stride of 1. This layer captures the relationship between two adjacent instructions. The instructions in positional feed-forward network (PFFN) in Transformer are arranged into two dimensions by setting the stride and kernel size to 1.
- The third, fourth and fifth layers correspond to attention, batch normalization and fully connected layers with `K` nodes as output classes to classify the network, respectively. RELU is the activation function in each intermediate layer, whereas the final layer uses softmax for classification. In the model's back-propagation algorithm, the stochastic gradient descent (SGD) method is used to minimize the error function [25].

3 Motivation

Our motivation for this study is the observation of alignment paddings and their placement in the binary with respect to different compilers and optimization settings. To motivate the idea of leveraging alignment padding for compiler provenance recovery, we analyze the binary code of the function `sqlite3_str_vappendf` compiled with three compilers and optimization settings, and see if the corresponding alignment padding presents unique patterns. Listing 1 shows different snippets of the binary code disassembled at the entry of a variadic function in *sqlite3* C application.

```
SQLITE_API void sqlite3_str_vappendf(sqlite3_str *pAccum, const char
    *fmt, va_list ap) {...}
```

From Listing 1, our first observation is that MSVC and Clangcl compilers prefer to insert `INT3` bytes at the entry of a function, whereas ICC emits `NOP` in-place of `INT3` for the function-entry alignment. In this particular example, the alignment padding at the function-entry in O2 and Ox does not differentiate from



MSVC (/Od)		
1	0x152f3	retq
2	**0x152f4**	**12x consecutive int3**
14	0x15300	mov %r8,0x18(%rsp)

MSVC (/O1)		
1	0xe69c	retq
2	**0xe69d**	**3x consecutive int3**
5	0xe6a0	mov %rsp,%rax

MSVC (/O2, /Ox)		
1	0x1a0d1	retq
2	**0x1a0d2**	**14x consecutive int3**
16	0x1a0e0	rex push %rbp

Clangcl (/Od)		
1	0x2f07	retq
2	**0x2f08**	**8x consecutive int3**
10	0x2f10	push %rsi

Clangcl (/O1)		
1	0x242c	jmpq 0x180002279
2	**0x2431**	**3x consecutive int3**
5	0x2434	push %r15

Clangcl (/O2, /Ox)		
1	0x360a	jmpq 0x180003410
2	**0x360f**	**int3**
3	0x3610	push %r15

ICC (/Od)		
1	0x8436	retq
2	**0x8437**	**nop**
3	0x8439	push %rbp

ICC (/O1)		
1	0x38f5	retq
2	**0x38f6**	**2x consecutive NOPs**
4	0x38f8	push %rbx

ICC (/O2, /Ox)		
1	0x7560	retq
2	**0x7561**	**nopl 0x0(%rax,%rax,1)**
3	**0x7568**	
4	**0x7569**	**nopl 0x0(%rax)**
5	0x7570	push %rbx

Listing 1 – Alignment padding at the entry of a variadic function *sqlite3_vtr_vappendf* in sqlite3 application.

each other. This is because the compilers will only generate different binary code in the corresponding optimization levels, when there exist duplicate copies of constant data elements and function definitions in the binary.[9] To demonstrate the frequency of alignment padding in binaries, we look at another example among the complex set of C projects in our database (i.e., *openssl*), as listed in Table 3. Here, we observe that the alignment padding in O2 and O3 vary in the MinGW compiler as compared to the other three compilers. This is because the MinGW compiler in O3 applies aggressive optimization strategies, like function inlining and loop unrolling, as compared to O2.

Moreover, the frequency of alignment in `.text` section highlights that `INT3` is the most frequently used alignment padding for the functions compiled with MSVC and Clangcl compilers. The reason is due to the fact that compilers emit `INT3` instruction as debugger trap to gracefully handle the execution in case of an exception (See Footnote 9). In contrast, MinGW compiler utilizes a larger number of `NOP`s for aligning the optimized instructions.

Interestingly, we found that `DB` and `ZERO` are frequently emitted alignment bytes in the data sections of compiled binaries at higher optimization levels. Therefore, to favor the small size of optimized binaries, compilers allocate data sections for alignment padding and emit reduced code in the `.text` section, respectively. Overall, all compilers emit frequent `ZERO` alignment padding in the data sections of the binaries, including the *end-of-section* alignment padding (See Footnote 9). Thus, we can say that there is a significant variation of alignment padding found in the binaries compiled with various compilers and at different optimization settings on Windows. Therefore, to achieve the best possible performance, it is a common practice for compilers to enforce natural alignment of both data and code [2,22]. However, the compiler's strategy for alignment

[9] https://docs.microsoft.com/en-us/cpp/build/reference/compiler-options-listed-alphabetically?view=msvc-160.

Table 3. Number of Alignment padding bytes in the `.text` and `.data` sections of *openssl* application, compiled with four compilers at four optimization levels.

Section		`.text`				`.data`			
Padding	Opt	MSVC	ICC	Clangcl	MinGW	MSVC	ICC	Clangcl	MinGW
#`NOP`	`Od/O0`	52	648	140	12,109	156	67	85	124
	`O1`	15	564	70	6,630	126	90	75	124
	`O2`	433	871	597	25,499	142	99	56	127
	`Ox/O3`	433	871	597	26,481	142	99	56	138
#`DB`	`Od/O0`	264	0	1,419	15	2,710	2,594	3,635	2,665
	`O1`	58	0	1,347	16	1,855	1,829	2,129	2,685
	`O2`	269	98	2,644	14	1,817	3,825	2,092	2,656
	`Ox/O3`	269	98	2,644	16	1,817	3,825	2,092	4,260
#`INT3`	`Od/O0`	3,438	117	15,146	0	47	48	55	64
	`O1`	671	144	12,156	0	49	66	50	66
	`O2`	2,608	188	14,125	0	58	53	62	51
	`Ox/O3`	2,608	188	14,125	0	58	53	62	69
#`ZERO`	`Od/O0`	164	0	2	9	11,726	15,388	14,710	17,477
	`O1`	150	0	6	9	13,646	16,422	13,322	17,492
	`O2`	222	9,682	4	9	12,454	18,714	11,120	17,283
	`Ox/O3`	222	9,682	4	9	12,454	18,714	11,120	16,263

padding varies from platform to platform. Thus, to compare Windows and Linux, Windows specify additional alignment options to align the sections and pages of PEs and DLLs on a 4K-byte boundary (See Footnote 9). Whereas, the sections on Linux are aligned on a 4-byte boundary [22,23]. These specifications are additional to the data alignment for optimized code constructs and transfer operations [22,35]. It is worth noting here that ELF x86-64 ABI does not require the virtual and physical address to be page-aligned. Though different from Linux, the alignment padding on Windows shows interesting patterns in the compiled binary, which is significant for compiler provenance. We thus highlight the importance of alignment padding on Windows and demonstrate that it is more useful for compiler provenance.

4 Compiler Provenance Recovery Model

In Sect. 3, we saw that different compilers and optimization levels emit unique signatures in the binary code compiled with different settings. In this section, we review the state-of-the-art deep learning model, o-glassesX and present our enhanced model, BinAlign. A deep neural network trained completely on data without domain knowledge might be non-explainable [38], whereas a system based entirely on expert knowledge may have limitations due to insufficient inference logic [28]. Xu et al. [38] state that adding node embeddings to control flow graph (CFG) of binary functions enhance the performance of the underlying binary similarity model. With this background, we improve the existing compiler provenance model, o-glassesX, and embed expert knowledge of alignment padding into the CNN based deep learning model.

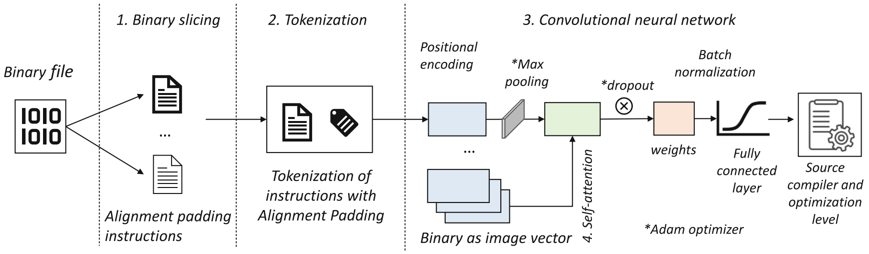


Fig. 1. Design overview of BinAlign and improved BinAlign architecture

4.1 BinAlign

The architecture of BinAlign is illustrated in Fig. 1. BinAlign follows the same approach as o-glassesX for classifying the binary instructions into different classes of compiler provenance. It is thus composed of the following three major components,[10] i.e., 1) binary slicing, 2) padding tokenization, 3) CNN. The neural network consists of the following core layers, i.e., i) positional encoding, ii) attention block, iii) batch normalization, and iv) fully connected layer.

The input to the network is a sequence of 128-bit fixed-length instructions that are embedded with alignment padding information. Thus, binary instructions are read in the form of image vector encodings. An image vector is a numerical representation of the binary in which each pixel is represented by a value of either 0 or 1.

To incorporate alignment padding in the underlying CNN architecture, we slice the binary code and detect patterns associated with alignment padding. We then categorize the alignment padding instructions into different groups and assign them the corresponding tokens (denoted by TOK), as shown at lines 6, 8, 10, and 12 of Algorithm 1. Thus, we encode the tokens along with the instructions and input the vectors into positional encoding layer preceding the attention layer of BinAlign. Following the approach in previous work [24], we pad all instructions with zeros to make them a uniform 16-byte length. Additionally, our approach considers all variations of the NOP instruction (as outlined in Table 4) to be incorporated into the model.

After tokenization, the positional encoding layer of BinAlign captures the relationship between two adjacent instructions. Thus, the positional encodings are learnt by the attention layer that utilizes self-attention to generate weights and focuses on a portion of the input information to classify binary sequences. Finally, the output of the attention block is passed through batch normalization layer to stabilize the network. Thus, the final layer of CNN (i.e., the fully connected layer) classifies the network into various output classes. RELU [25] is the activation function in each intermediate layer, whereas the final layer uses softmax for classification. In the model's back-propagation algorithm, the stochastic gradient descent (SGD) [25] method is used to minimize the error function.

[10] The improvements (i.e., additional layers and optimization algorithm) marked with * in Fig. 1, belong to the improved BinAlign design.

Algorithm 1. Binary Slicing and Instruction Tokenization

```
procedure FetchAlignmentPaddings(binary)
    foreach Insn in binaryCode
    if instruction == AlignmentPadding then
        Do foreach {Insn} in the AlignmentPadding
        if $opcode_{Insn}$ == DATA then
            Assign TOK of "DB" instructions
        else if $opcode_{Insn}$ == ZERO then
            Assign TOK of "ZERO" instructions
        else if $opcode_{Insn}$ == INT3 then
            Assign TOK of "CC" instructions
        else if $opcode_{Insn}$ == NOP then
            Assign TOK of "NOP" instructions
    else if instruction ∈ Filler then
        Assign TOK of "Filler" instructions
    else if instruction ∈ non-Alignment then
        Assign TOK of "non-padding" instruction
```

Table 4. Multi-byte NOP alignment padding

Opcode	Operand	no. of Bytes	Hex representation
NOP	<no operand>	1	90
NOP	<no operand>	2	6690
NOP	WORD [RAX+RAX+0x0]	9	660f1f840000000000
NOP	WORD [RAX+RAX+0x0]	10	662e0f1f840000000000
NOP	DWORD [RAX]	3	0f1f00
NOP	DWORD [RAX+0x0]	4	0f1f4000
NOP	DWORD [RAX+RAX+0x0]	5	0f1f440000

4.2 Improved BinAlign

Due to our added attributes in the form of alignment padding tokens (see Fig. 1) in the existing model [25], we need to enhance the underlying architecture with additional layers. This is because the alignment padding is not uniformly distributed in the binary. For example, at one point the compiler may align the end-of-section with a large number of recurring padding bytes, whereas at another location, a multi-byte instruction may be generated to enforce the instruction alignment. In Table 5, we present nine scenarios in which the compilers emit alignment padding at different locations in the binary. Therefore, it becomes necessary for BinAlign to include additional layers in the underlying architecture to enhance its performance. Thus, we add maxpooling, dropout and adam optimizer to enhance the basic architecture of BinAlign and name it as the improved BinAlign as shown in Fig. 1. We briefly explain the additional layers and optimization in this section.

Table 5. Nine scenarios when compilers emits alignment padding at different locations in the binary with respect to optimization

#	Placement in Binary	Purpose and Location	Impact of Optimization	Align type
1	Data interleaved in Code	inline data in .text section	increase with optimization	`DB`
2	Import Functions	before a branch instruction	decrease with optimization	`NOP`
3	Exception handling (EH) functions	aligned jump tables	less in MSVC & Clangcl, intense in ICC at O2, Ox[a]	`NOP`
4	Intrinsic functions	compiler's inlined functions	expansion of function vary with compiler family & optimization [19]	`NOP`, `DB`
5	Function-entry alignment	before subroutine or EH function	align code and data	`INT3`, `NOP`
6	Common Runtime (CRT) routines	handcoded assembly routines	intense use of CRT at higher optimization	`ZERO`, `DB`
7	End-of-segment alignment	PE sections are aligned	decrease with higher optimization	`ZERO`, `DB`
8	Vector operations	128-bit multimedia operands are aligned	`MMX` and `SSE (XMM)` instructions aligned at 16 Byte address	`NOP`, `DB`
9	Branch Alignment	align branch target to a multiple of 16	increase with optimization	`NOP`

[a] https://support.alfasoft.com/hc/en-us/articles/360002874938 (Intel compatibility with MSVC).

(1) *Pooling* [15] is a regularization technique that reduces overfitting, whereas *max pooling* decreases the computational cost by reducing the number of parameters to learn the features. We add a max-pooling layer after the convolution layer to extract shallow features from binary code with `K` as `96`, and stride length as `128`. Thus, pooling assists deep learning architectures to reduce computational cost caused by the dimensionality reduction problem [15].

(2) *Dropout* [15] assists the neural network in achieving high-quality performance on the test set and prevents the model from overfitting. We apply dropout to the fully connected layer of the neural network. Thus, by utilizing max-pooling and dropout, we aim to achieve the stochastic pooling in terms of activation picking, inspired by dropout regularization approach [15]. Here, we add a dropout layer with a rate of 0.5.

(3) *Adam optimization* [13] is an extension to classical stochastic gradient descent (SGD) to update network weights iteratively based on training data. SGD algorithm maintains a single learning rate (i.e., α) for weight updates which does not change during training. It has two more extensions, i.e., AdaGrad [41] and RMSProp [41] and Adam optimizer combines the benefits of both extensions of SGD. Thus, in order to optimize for better accuracy,

we replace SGD with Adam optimizer having following parameters, i.e., a) α as the coefficient for learning rate is set to 0.001, b) $\beta 1$ and $\beta 2$ as the exponential decay rates are set to 0.9 and 0.999, respectively, and c) ϵ is initialized as 1e−08. Thus, we train the improved BinAlign model over all the unoptimized and optimized binaries to learn the source compiler and optimization level classification. Similar to o-glassesX, we parse the binaries with distorm3[11] disassembler and get x86-64 assembly instructions.

5 Experimental Design

In this section, we explain our experimental design to evaluate the precision (P), recall (R) and f-measure (F1) of inferring the source compiler, and optimization level for o-glassesX, BinAlign (i.e., state-of-the-art with alignment padding), and improved BinAlign (i.e., state-of-the-art with alignment padding and additional layers in the CNN architecture). Hence, our evaluation is performed on a server machine having Ubuntu 22.04.1 LTS OS with 3.5 GHz and upto 64 CPU core processors with 132 GB RAM and 4 GeForce RTX2080 Ti GPUs having 12GB of memory. We thus perform experimentation on a large collection of binaries compiled with latest compilers on Windows, as they implement modern compiler optimization strategies. Here it is worth mentioning that we are not replicating the experiments in the o-glassesX paper. We therefore, focus on compiler provenance inference in x86-64 architecture for benign code and x86 for malicious code. The dataset is compiled from a set of 457 well-known C/C++ open-source projects using four commonly used compilers, i.e., MSVC-19, Clangcl-10, ICC-2021 and MinGW-10.

For the ground truth, we record the source compiler and optimization level label and compile all application in release build with debugging symbols enabled. Since, the debug information is present in separate PDB files that provide us information about the function-entry addresses, we study the differences in the alignment padding at the function start locations using the information extracted from the symbols. However, it is worth noting here that all tasks of provenance recovery training and inference are performed on the stripped binaries.

We therefore generate a diverse set of binaries that range in size from a few kilobytes to 40 megabytes. For each project, we build dependent libraries and programs separately. Further details regarding some of the projects in our dataset can be found in Table 6. Hence, we publish our compiled dataset and program source code for further study and research[12] (Table 7).

[11] https://pypi.org/project/distorm3.

[12] https://github.com/mali-arf/binalign_comp.

Table 6. The selected C/C++ projects from Github in our dataset

Project	Description
ogg-vorbis	audio encoder/decoder for lossy compression
cmake	a cross-platform, open-source build system generator
curl	library for data transfer with url syntax
doxygen	document generation tool from annotated C++ sources
eigen	C++ template library for linear algebra, matrices, vectors, numerical solvers, etc.
gflags	C++ library with command-line flags for strings, etc.
glm	C++ openGL Mathematics library for graphics
glog	C++ google logging (glog) module
libevent	a library that provides asynchronous event notifications
llvm	an open source compiler and toolchain
onednn	oneAPI deep neural network library optimized for Intel and Xe architectures
opencl	Open Computing Language for heterogeneous platforms like CPUs, GPUs, etc.
openssl	library to secure communications over computer networks against eavesdropping
Microsoft lib	C++ Standard Template Library (STL) for MSVC toolset and Visual Studio IDE
sqlite	a relational database management system contained in a C library
vtk	an image processing, 3D graphics, volume rendering, and visualization toolkit
zlib	data compression library used in gzip file compression programs

Table 7. The number of compiled binaries in our dataset

Compiler	Version	number of binaries			
		Od/OO	O1	O2	Ox/O3
MSVC	19.28.29	28,359	23,201	21,982	32,705
Clangcl	10.0.0	18,834	9,938	11,512	19,354
ICC	2021.1.2	14,475	10,193	11,816	14,798
MinGW	10.3.0	22,606	17,573	16,325	22,498
Total		84,274	60,905	61,635	89,355

We collect C/C++ applications in our dataset comprising of a large number of stars and widely used in binary analysis research. We, then construct a dataset of 296,169 binaries compiled with four major compiler optimization settings (see Sect. 5).

Thus, we fine-tune the evaluation models to get best results from o-glassesX, BinAlign and Improved BinAlign. o-glassesX utilizes four main hyperparameters that are tuned to achieve the best results in each model. Therefore, all models achieve the best average accuracy at the same hyperparameter setting with i) *batch size* as 1000, ii) *instruction length* as 64, iii) *sample size* as 100,000, and, iv) *epoch* as 50. Hence, we used 4-fold cross-validation to perform training and testing of data for four tasks, i.e., source compiler inference, compiler optimization inference, and joint source compiler with two and four levels of compiler optimization inference, respectively. The average results on the compiler provenance tasks are shown in Tables 8, 9, 10 and 11.

6 Results

In this section, we present our experimental results in the form of answers to the RQs discussed in Sect. 1. To ensure a fair comparison with previous research, we calculate the average test results for each model in order to draw conclusions about the source compiler, optimization levels, and joint source compiler and optimization levels.

6.1 RQ1. How Effective is BinAlign in Identifying the Source Compiler of Binaries?

Table 8 measures our models' recall (R), precision (P) and f-measure (F1) while inferring the source compiler for the three models. Overall, our improved BinAlign model shows increased performance while inferring the source compiler of the test binaries. One of the reasons for improved performance on MSVC compiled binaries is the inference of `NOP` bytes to align the data for compiler-specific intrinsic functions. Which the model learns to correctly infer the compiler specific alignment. Similarly, for ICC compiled binaries, the improved BinAlign performs much better than o-glassesX for classifying the NOP bytes embedded before every `CALL` to the internal compiler functions.

6.2 RQ2. How Effective is BinAlign in Identifying the Compiler Optimization Level of a Binary?

Tables 9a and 9b present the results on the classification of the two- and four-level of compiler optimizations, respectively. The results of the two-level optimization inference task, as presented in Table 9a, demonstrate that the improved BinAlign model can accurately determine the optimization level of the most optimized binaries, with a level of precision that is almost as high as that for the unoptimized binaries. However, Table 9b shows that the least correctly inferred compiler optimization setting for BinAlign is O2. This is because of an optimization strategy that is followed by the compilers to maintain a single copy for the common data and functions in the binary (i.e., COMDAT (See Footnote 9)). The optimization confines the alignment padding, thereby affecting the inference of the highly optimized binaries.

However, the improved BinAlign demonstrates a relatively higher accuracy in inferring class labels for O1 and Ox optimized binaries. This can be attributed to the optimization strategy employed by compilers on Windows, which generates separate copies of aligned functions and data with multiple definitions at the Ox optimization level. Conversely, the code generated by the O1 optimization level remains consistent across all compilers. This consistency arises because compilers tend to reserve most of the alignment padding in the data sections, which favors the reduced size of the binary code. Overall, our evaluation results show that optimizations performed by compilers generate variant and complex code, making it difficult for the compiler prediction neural network based models to learn the consistent patterns. However, the improved BinAlign significantly improves the existing state-of-the-art model in recovering the compiler optimization inference of real-world binaries.

Table 8. The result of source compiler inference

Compiler	o-glassesX			BinAlign			Improved BinAlign		
	R	P	F1	R	P	F1	R	P	F1
MSVC	0.9729	0.9797	0.9763	0.9953	0.9934	0.9943	0.9938	0.9972	0.9955
Clangcl	0.9465	0.9470	0.9467	0.9787	0.9809	0.9798	0.9739	0.9890	0.9814
ICC	0.9662	0.9697	0.9679	0.9890	0.9907	0.9899	0.9944	0.9862	0.9903
MinGW	0.9216	0.8988	0.9100	0.9643	0.9613	0.9628	0.9812	0.9646	0.9728
Overall	0.9576	0.9576	0.9576	0.9850	0.9850	0.9850	0.9873	0.9873	0.9873

Table 9. The result of two and four levels of compiler optimization level inference

(a) Two-level compiler optimization inference (i.e., Od/O0 and Ox/O3)

	o-glassesX			BinAlign			Improved BinAlign		
Opt	R	P	F1	R	P	F1	R	P	F1
Od/O0	0.9783	0.9736	0.9764	0.9888	0.9896	0.9892	0.9914	0.9902	0.9908
Ox/O3	0.9696	0.9750	0.9723	0.9883	0.9874	0.9879	0.9890	0.9903	0.9897
Overall	0.9739	0.9743	0.9743	0.9886	0.9886	0.9886	0.9902	0.9902	0.9902

(b) Four-level compiler optimization inference (i.e., Od/O0, O1, O2, Ox/O3)

	o-glassesX			BinAlign			Improved BinAlign		
Opt	R	P	F1	R	P	F1	R	P	F1
Od/O0	0.9839	0.9838	0.9838	0.9873	0.9885	0.9879	0.9915	0.9874	0.9894
O1	0.7365	0.7275	0.7320	0.7313	0.8078	0.7676	0.8002	0.8375	0.8184
O2	0.6971	0.6011	0.6456	0.6316	0.6701	0.6503	0.8248	0.8217	0.8232
Ox/O3	0.6371	0.6276	0.6323	0.7198	0.8714	0.7884	0.7495	0.9083	0.8213
Overall	0.7678	0.7472	0.7562	0.7730	0.8298	0.7995	0.8318	0.8811	0.8547

Table 10. Inference of joint source compiler with two levels of compiler optimization

Compiler-opt	o-glassesX			BinAlign			Improved BinAlign		
	R	P	F1	R	P	F1	R	P	F1
MSVC-Od	0.9667	0.9643	0.9655	0.9886	0.9743	0.9814	0.9991	0.9925	0.9958
MSVC-Ox	0.9357	0.9535	0.9445	0.9651	0.9795	0.9722	0.9735	0.9808	0.9772
Clangcl-Od	0.9779	0.9744	0.9761	0.9885	0.9888	0.9886	0.9985	0.9998	0.9992
Clangcl-Ox	0.9088	0.9212	0.9150	0.9599	0.9508	0.9553	0.9508	0.9618	0.9563
ICC-Od	0.9831	0.9905	0.9868	0.9945	0.9971	0.9958	0.9932	0.9995	0.9963
ICC-Ox	0.9519	0.9423	0.9471	0.9777	0.9840	0.9808	0.9732	0.9815	0.9773
MinGW-O0	0.9711	0.9668	0.9690	0.9949	0.9868	0.9908	0.9993	0.9942	0.9967
MinGW-O3	0.8438	0.7975	0.8200	0.9085	0.9164	0.9124	0.9367	0.9194	0.9279
Overall	0.9515	0.9515	0.9515	0.9772	0.9772	0.9772	0.9781	0.9787	0.9785

Table 11. Inference of joint source compiler with four levels of compiler optimization

Compile-opt	o-glassesX			BinAlign			Improved BinAlign		
	R	P	F1	R	P	F1	R	P	F1
MSVC-Od	0.9715	0.9695	0.9705	0.9874	0.9718	0.9795	0.9826	0.9737	0.9781
MSVC-O1	0.7692	0.6615	0.7113	0.7732	0.7961	0.7845	0.8880	0.7986	0.8409
MSVC-O2	0.4995	0.2488	0.3322	0.5806	0.0845	0.1476	0.5223	0.3005	0.3816
MSVC-Ox	0.7109	0.8602	0.7784	0.7113	0.9127	0.7995	0.7458	0.8791	0.8070
Clangcl-Od	0.9810	0.9808	0.9809	0.9883	0.9850	0.9866	0.9932	0.9816	0.9874
Clangcl-O1	0.6215	0.4529	0.5240	0.7532	0.5485	0.6348	0.7721	0.6354	0.6971
Clangcl-O2	0.3373	0.0696	0.1154	0.6477	0.5503	0.5951	0.6515	0.7314	0.6892
Clangcl-Ox	0.7392	0.8892	0.8073	0.6126	0.7292	0.6658	0.7023	0.6501	0.6752
ICC-Od	0.9888	0.9929	0.9909	0.9950	0.9943	0.9947	0.9941	0.9979	0.9960
ICC-O1	0.7487	0.5993	0.6657	0.7614	0.7666	0.7640	0.8759	0.8062	0.8396
ICC-O2	0.3853	0.1187	0.1815	0.6586	0.0890	0.1568	0.4455	0.2701	0.3363
ICC-Ox	0.7158	0.8959	0.7958	0.7243	0.9367	0.8169	0.7468	0.8624	0.8005
MinGW-O0	0.9782	0.9764	0.9773	0.9833	0.9917	0.9875	0.9928	0.9868	0.9898
MinGW-O1	0.9466	0.9372	0.9419	0.9736	0.9464	0.9598	0.9640	0.9678	0.9659
MinGW-O2	0.6356	0.6345	0.6351	0.6123	0.7067	0.6561	0.7051	0.7541	0.7287
MinGW-O3	0.6363	0.6250	0.6306	0.6373	0.5674	0.6003	0.7715	0.6957	0.7316
Overall	0.7422	0.6979	0.7053	0.7869	0.7385	0.7357	0.8077	0.7806	0.7896

6.3 RQ3. How Effective is BinAlign in Identifying the Joint Source Compiler and Optimization Level of a Windows Binary?

Tables 10 and 11 show the performance of three compiler provenance models, while predicting the joint source compiler and optimization level of the compiled binaries with two and four levels of optimization, respectively. For the two-level joint source compiler and optimization level inference as shown in Table 10, our improved BinAlign performs the best on compiler provenance recovery of MSVC and ICC compiled binaries with better inference on unoptimized binaries as compared to the highly optimized ones.

Moreover, from the results we can see that the neural network models do not perform perfectly well, while inferring the joint source compiler and optimization level, specifically for MSVC and ICC compiled test binaries, at O2 optimization level. This is because, one of the challenges for deep learning model is the highly optimized code and the inter-procedural optimization in optimized binaries.

Furthermore, it is observed that neural network models perform relatively better when the binaries are compiled with either unoptimized settings or with the most-aggressive optimization settings using MSVC and ICC compilers. This may be attributed to the fact that modern C/C++ compilers provide support for advanced vector extension (AVX) architecture, which generates over-aligned instructions, thus leading to improved performance (See Footnote 5). Also, our test set instances comprise of instructions operating on floating point data, for which the ICC compiler emits alignment padding in the form of `DB` bytes to align the instructions and data for vector operations.

On the other hand, for unoptimized binaries, the improved BinAlign model learns the alignment padding patterns comparatively well, as the instructions are padded with `DB 90` and `DB 0CC` in the `.text` section of the compiled binaries, whereas `DB 0` in the data sections, respectively. One of the significant characteristics of the deep learning CNN models is their higher performance and better accuracy for the zero-padded data bytes [15]. Hence, from our evaluation results of joint prediction of source compiler and optimization level, we conclude that the improved BinAlign recovers the compiler provenance of Clangcl, and MSVC compiled binaries with a relatively better score as compared to other compilers.

6.4 Malware Case Study

To illustrate the generality of our approach for compiler provenance inference of real-world malware, we compile 113 C/C++ source code of Win32 malware downloaded from theZoo[13] repository. Since the malware source uses the core Windows APIs which are incompatible with Clangcl, ICC and MinGW compilers, we therefore compile the projects on three different versions of MSVC compiler, i.e., VS2015, VS2017 and VS2019, at four different optimization levels. Thus, we train our models on 64-bit benign programs and 32-bit malicious programs and test the models on malicious binaries only. The benign programs belong to the same set of programs as mentioned in Sect. 5, however for the current evaluation we compile them with MSVC compiler and perform testing with three different versions of the MSVC compiler.

Tables 12 and 13 present the malware families and the distribution of our malware dataset, respectively. In our dataset, we gather seven different families of malware programs in C/C++ language as shown in Table 12. The malware programs are then compiled with three versions of MSVC compiler in four different optimization settings as shown in Table 13, with the most successful compilation in version 2019 that supports the most APIs. Table 14a shows the overall classification result of compiler version prediction and compiler optimization prediction of our malware dataset. Hence, our results demonstrate that the improved BinAlign outperforms other models in predicting the compiler version of malware binaries. This is because adversaries may introduce binary padding to add junk data in the code and change the on-disk representation of the binaries. Here, we acknowledge that despite the obfuscation implemented in a smaller set of malicious binaries as compared to a larger set of benign programs, our improved model is able to predict compiler version and optimization level with a promising score.

[13] https://github.com/ytisf/theZoo.

Table 12. The number and types of binaries belonging to different families of malware.

Family	Malware Type	#bins
Dexter	Point of Sales Trojan	1
Rovnix	Bootkit	1
Carberp	Botnet	36
BJWJ	Banking Trojan	6
Anti_Rapport	Banking Trojan	11
Trochilus	Remote Access Trojan	40
ZeroAccess	Rootkit	18
Total		113

Table 13. The number of malware binaries successfully compiled with three different versions of MSVC compiler and at four different optimization levels.

Opt	VS2019	VS2017	VS2015	Total
`Od`	17	8	8	33
`O1`	13	5	5	23
`O2`	21	10	4	35
`Ox`	14	4	4	22
Total	65	27	21	113

Table 14. Compiler, version and optimization level inference of malware and custom aligned binaries

(a) Compiler version and optimization level inference of malware binaries

	Version			Opt		
Metrics	o-glassesX	BinAlign	Improved BinAlign	o-glassesX	BinAlign	Improved BinAlign
P	0.8540	0.8713	0.9160	0.8718	0.8825	0.9287
R	0.8548	0.8729	0.9170	0.8740	0.8827	0.9295
F1	0.8542	0.8719	0.9162	0.8722	0.8822	0.9290

(b) Compiler and optimization level inference of custom-aligned binaries

	Compiler			Opt		
Metrics	o-glassesX	BinAlign	Improved BinAlign	o-glassesX	BinAlign	Improved BinAlign
P	0.7923	0.8280	0.8557	0.6892	0.7708	0.7950
R	0.7927	0.8294	0.8548	0.6729	0.7709	0.7920
F1	0.7925	0.8287	0.8552	0.6811	0.7709	0.7935

6.5 Custom Alignment Padding

Considering a scenario when software developers or malware authors intentionally modify the compiler's default alignment settings, we conduct a case study comprising the binaries that enforce custom alignment padding in the compiled binaries. For that, we perform an extensive study of the compiler options that support aligning data within sections, structures, data packing and section alignment. Hence, we found that MSVC (See Footnote 9), Clangcl and ICC support four major alignment settings, defined as, i) `ALIGN`, ii) `FILEALIGN`, iii) `Zc:alignedNew`, and iv) `Zp16`. These options correspond to the section alignment in linear address space, alignment of sections to the output file, alignment of dynamically-allocated data and the packing of structure member alignment, respectively. On the other hand, MinGW compiler supports alignment of c++17 data standard, aligning data for functions, labels, jumps, and loops, respectively. Here, the compiler options to achieve the respective alignment is `-faligned-new`, `-falign-functions`, `-falign-jumps`, `-falign-labels`, and `-falign-loops`, respectively [2]. Thus, we train the models with custom alignment of 8192 bytes for all the alignment options discussed above.

Therefore, to evaluate our models on custom-aligned compiled binaries, we train the compiler provenance models with, i) default alignment compiler settings, and ii) custom-alignment padding on the same dataset as evaluated in Sect. 6.1 to 6.3. We then test our trained models with custom-aligned binary code to measure their evaluation performance. Therefore, we utilize 70% of our total compiled data for training, while 30% of the custom-aligned binaries for testing. Hence, our evaluation results show that the improved BinAlign infers the source compiler and compiler optimization level of custom-aligned compiled binaries with a fairly decent score as shown in Table 14b.

7 Limitation

We evaluate BinAlign only on the selected options of custom alignment padding provided by the compiler. However, the alignment options in MinGW compiler (for example) for aligning data in C++17 standard, functions, loops, jumps and labels have a long range from 2^2 to 2^{16}. Hence, it would be interesting to assign labels to the differently padded binary code with varying alignment padding settings. We thus leave this as the future work. Additionally, this work can further be extended to incorporate the architecture-specific alignment padding [2].

8 Conclusion

In this work, we evaluate the state-of-the-art compiler provenance recovery model, o-glassesX and propose BinAlign that incorporates compiler-optimization-specific domain knowledge into the existing model. Thus, by learning the alignment padding in the binary code, our model infers the source compiler and optimization level over a diverse set of real-world binaries including 64-bit benign and 32-bit malware binaries. We thus believe that newer patterns of instructions for alignment padding may be explored to enhance the compiler provenance analysis. Moreover, this work can be extended to incorporate additional filler instructions to enhance the performance of compiler provenance analysis in optimized binaries.

Finally, we sincerely thank the authors of o-glassesX (Otsubo et al.) for providing us with the replication package along with the necessary guidance.

References

1. Microsoft Visual C++. https://visualstudio.microsoft.com/vs/features/cplusplus/. Accessed 23 Apr 2023
2. MinGW-w64 compiler (2023). https://www.mingw-w64.org/. Accessed 20 Apr 2023

3. Andriesse, D.: Practical Binary Analysis: Build Your Own Linux Tools for Binary Instrumentation, Analysis, and Disassembly (2018). https://books.google.com.sg/books?id=laWgswEACAAJ
4. Andriesse, D., Chen, X., van der Veen, V., Slowinska, A., Bos, H.: An in-depth analysis of disassembly on full-scale x86/x64 binaries. In: 25th USENIX Security Symposium (2016)
5. Benoit, T., Marion, J.Y., Bardin, S.: Binary level toolchain provenance identification with graph neural networks. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) (2021)
6. Chaki, S., Cohen, C., Gurfinkel, A.: Supervised learning for provenance-similarity of binaries. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 15–23 (2011)
7. Cifuentes, C., Gough, K.J.: Decompilation of binary programs. Softw. Pract. Experience **25**(7), 811–829 (1995)
8. Ding, S.H., Fung, B.C., Charland, P.: Asm2Vec: boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 472–489. IEEE (2019)
9. Eagle, C.: The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler. No Starch Press, USA (2011)
10. Grune, D., Van Reeuwijk, K., Bal, H.E., Jacobs, C.J., Langendoen, K.: Modern Compiler Design. Springer, Heidelberg (2012)
11. Ji, Y., Cui, L., Huang, H.H.: BugGraph: differentiating source-binary code similarity with graph triplet-loss network, pp. 702–715. ACM, New York (2021)
12. Ji, Y., Cui, L., Huang, H.H.: Vestige: identifying binary code provenance for vulnerability detection. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 2021. LNCS, vol. 12727, pp. 287–310. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78375-4_12
13. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
14. Koo, H., Park, S., Choi, D., Kim, T.: Semantic-aware binary code representation with BERT. arXiv preprint arXiv:2106.05478 (2021)
15. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017)
16. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 207–226. Springer, Heidelberg (2006). https://doi.org/10.1007/11663812_11
17. Lin, Y., Gao, D.: When function signature recovery meets compiler optimization. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 36–52. IEEE (2021)
18. Lin, Y., Gao, D., Lo, D.: Resil: revivifying function signature inference using deep learning with domain-specific knowledge. In: Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy, pp. 107–118 (2022)
19. Lopes, B.C., Auler, R.: LLVM: Building a Modern Compiler Infrastructure (2020)
20. Massarelli, L., Di Luna, G.A., Petroni, F., Querzoni, L., Baldoni, R.: Investigating graph embedding neural networks with unsupervised features extraction for binary analysis. In: Proceedings of the 2nd Workshop on Binary Analysis Research (BAR) (2019)
21. Massarelli, L., Di Luna, G.A., Petroni, F., Baldoni, R., Querzoni, L.: SAFE: self-attentive function embeddings for binary similarity. In: Perdisci, R., Maurice, C.,

Giacinto, G., Almgren, M. (eds.) DIMVA 2019. LNCS, vol. 11543, pp. 309–329. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22038-9_15

22. Mitchell, M., Oldham, J., Samuel, A.: Advanced Linux Programming. New Riders, Berkeley (2001)
23. Muchnick, S., et al.: Advanced Compiler Design Implementation. Morgan Kaufmann (1997)
24. Otsubo, Y., Otsuka, A., Mimura, M., Sakaki, T.: o-glasses: visualizing x86 code from binary using a 1D-CNN. IEEE Access **8**, 31753–31763 (2020)
25. Otsubo, Y., Otsuka, A., Mimura, M., Sakaki, T., Ukegawa, H.: o-glassesx: compiler provenance recovery with attention mechanism from a short code fragment. In: Proceedings of the 3rd Workshop on Binary Analysis Research (2020)
26. Pizzolotto, D., Inoue, K.: Identifying compiler and optimization options from binary code using deep learning approaches. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME) (2020)
27. Rahimian, A., Nouh, L., Mouheb, D., Huang, H.: Binary code fingerprinting for cybersecurity
28. Rahimian, A., Shirani, P., Alrbaee, S., Wang, L., Debbabi, M.: Bincomp: A stratified approach to compiler provenance attribution. In: Digital Investigation, the Proceedings of the Fifteenth Annual DFRWS Conference (2015)
29. Ramshaw, M.J.: Establishing malware attribution and binary provenance using multicompilation techniques (2017). https://www.osti.gov/biblio/1390004
30. Rosenblum, N., Miller, B.P., Zhu, X.: Recovering the toolchain provenance of binary code. In: Proceedings of the 2011 International Symposium on Software Testing and Analysis, ISSTA 2011 (2011)
31. Rosenblum, N.E., Miller, B.P., Zhu, X.: Extracting compiler provenance from program binaries. In: Proceedings of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, PASTE 2010 (2010)
32. Rosenblum, N.E., Zhu, X., Miller, B.P., Hunt, K.: Learning to analyze binary computer code. In: AAAI, pp. 798–804 (2008)
33. Shirani, P., Wang, L., Debbabi, M.: Binshape: scalable and robust binary library function identification using function shape. In: Detection of Intrusions and Malware, and Vulnerability Assessment (2017)
34. Tian, Z., Huang, Y., Xie, B., Chen, Y., Chen, L., Wu, D.: Fine-grained compiler identification with sequence-oriented neural modeling. IEEE Access **9**, 49160–49175 (2021)
35. TIS Committee: Executable and Linking Format (ELF) Specification (1995). https://refspecs.linuxfoundation/elf/elf.pdf
36. Wang, R., et al.: RAMBLR: making reassembly great again. In: NDSS (2017)
37. Wang, S., Wang, P., Wu, D.: Reassembleable disassembling. In: 24th USENIX Security Symposium, Washington D.C., pp. 627–642 (2015)
38. Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., Song, D.: Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proceedings of the 2017 ACM SIGSAC, pp. 363–376 (2017)
39. Xue, H., Sun, S., Venkataramani, G., Lan, T.: Machine learning-based analysis of program binaries: a comprehensive study. IEEE Access **7**, 65889–65912 (2019)
40. Yang, S., Shi, Z., Zhang, G., Li, M., Ma, Y., Sun, L.: Understand code style: efficient CNN-based compiler optimization recognition system. In: 2019 IEEE International Conference on Communications (ICC), ICC 2019 (2019)
41. Zou, F., Shen, L., Jie, Z., Zhang, W., Liu, W.: A sufficient condition for convergences of Adam and RMSProp. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11127–11135 (2019)

Encrypted Network Traffic Classification with Higher Order Graph Neural Network

Zulu Okonkwo(✉), Ernest Foo, Zhe Hou, Qinyi Li, and Zahra Jadidi

Griffith University, Gold Cost, Australia
{z.okonkwo,e.foo,z.hou,qinyi.li,z.jadidi}@griffith.edu.au

Abstract. Encryption protects internet users' data security and privacy but makes network traffic classification a much harder problem. Network traffic classification is essential for identifying and predicting user behaviour which is important for the overall task of network management. Deep learning methods used to tackle this problem have produced promising results. However, the conditions on which these experiments are carried out raise questions about their effectiveness when applied in the real world. We tackle this problem by modelling network traffic as graphs and applying deep learning for classification. We design a graph classifier based on higher order graph neural network with the aim of optimum generalisation. To demonstrate the robustness of our model, we cross validate it on the ISCXVPN and USTC-TFC datasets with varying input specifications. We use our model to demonstrate the impact of network data truncation on traffic classification and define benchmarks for input specifications. Our best results outperform the state-of-the-art in terms of generalisation strength. Our tool is available online (https://github.com/zuluokonkwo/Encrypted-Network-Traffic-Classification-with-Higher-Order-Graph-Neural-Network).

Keywords: Graph Neural Network · Encrypted Network Traffic · Classification · Network Security · Deep Learning

1 Introduction

In the design of network security systems, the core aim is an accurate classification of network traffic by the security apparatus [2]. Traditional network security involves combining various layers of security in a defence-in-depth approach. Each layer plays a role with the overall aim of controlling what data enters or exits the network. Encryption aids security by strengthening privacy amongst communicating channels, leading to its wide adoption across the internet. Over 94% of Google's traffic now runs over TLS or SSL [3]. While improving privacy over the internet, encryption has created new attack surfaces for bad actors to traverse unnoticed in and out of networks. Due to the sophisticated nature of encryption algorithms, security systems struggle to accurately classify network traffic. Recently, a new wave of encrypted attacks has spread across the internet,

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 630–650, 2023.
https://doi.org/10.1007/978-3-031-35486-1_27

with the rise of cyber physical systems(CPS) and internet of things (IoT) cyber threats over encrypted channels saw a 132% increase in 2022, and malware over HTTPS continues to rise [1].

Machine learning (ML) and Deep Learning (DL) based methods have experienced huge success in the overall task of network traffic classification. However, their results are questionable because of underlining issues. State-of-the-art ML based classifiers [2,4–6] have more emphasis on the model design stage while testing on a small set of data and not paying much attention to the data processing/feature extraction stage. This process yields good but questionable results as the generalisation capability of the model isn't guaranteed. State-of-the-art Graph neural networks (GNN) based designs [15,22] are built with low-order GNNs. This method utilises node locality for classification tasks, not taking advantage of higher-order information that has proven to preserve structural embeddings necessary for graph classification.

The imbalanced nature of network traffic datasets is a concern for ML/DL based classifiers; training with imbalanced data directly impacts a model's performance. Results produced with such datasets are biased towards the majority classes. State-of-the-art classifiers [4–6] utilised imbalanced datasets for their training process not clearly stating how this issue is contained. DL/ML classifiers deal with a lot of parameters that increase during model training to enable the network generalise optimally. An increase in parameters also accounts for an increased tendency of an over-fitting model. State-of-the-art classifiers [4–6,22] use dropout to deal with this tendency. Research by Garbin et al. [8] advised that dropouts be used with caution and when in doubt batch normalisation be used instead. Garbin et al. [8] also advised that dropouts be used with different rates to find the optimum spot which yields the best accuracy. Cross-validating and training a model across a range of datasets is a way to show a model is not overfitting. Network traffic is dynamic and varies in size. The number and sizes of packets in a traffic session vary for different applications. For ML/DL classification approaches, defining input specifications is a requirement for models. Therefore, defining input sizes that yield the best results when variable-size inputs cannot be used is important. State-of-the-art classifiers [4,6] truncate network input during training. A trade-off of such should be backed with a clear explanation or leverage DL methods that can perform computation on variable-sized inputs.

In this paper, we design an encrypted traffic classifier based on a higher-order graph neural network. Our design is based on a GNN blueprint by Morris et al. [21] that preserves structural information necessary for graph classification. We test our model on two datasets with different input specifications to investigate which is optimum for the task of encrypted traffic classification. Our model can be applied to the network layer of CPS, IoT and smart critical infrastructures to identify traffic that pose risks to the systems. In our training phase, we implement a weighted random sampler that ensures every class is well represented, making our model unbiased towards any class. We perform stratified cross validation across our dataset with a fold of 5 to ensure our results are as realistic as possible. Our model demonstrated better generalisation strength when compared to state-of-the-art that used the same dataset. Our best results for the VPN-dataset are above 97% across all evaluation metrics which is better than the compared literature. For the non-VPN dataset our best results are above 87% for all evaluation metrics. For the Benign and Malware classes of the USTC-TFC dataset, we achieve well over 95% for all evaluation metrics. Our contributions can be itemized as follows:

- Development of a higher order GNN-based model that can identify and classify encrypted network traffic with optimum generalisation strength.
- Improved the feature extraction process for encrypted traffic to better represent and show relationship of traffic flows.
- Evaluate the impacts of data truncation and padding for encrypted network traffic classification at the data-link layer (L2) and sessions layer (L5).

The rest of this paper is structured as follows: In Sect. 2, we introduce the preliminaries that lay foundations to our data processing and model design. In Sect. 3, we introduce our GNN-based classifier and define the data processing methodology. In Sect. 4, we evaluate our model on the ISCXVPN and USTC-TFC datasets. We also discuss the results gotten from our experiments and conduct a comparative analysis with state-of-the-art models. In Sect. 5, we highlight some related papers and discuss gaps which our paper addressed. In Sect. 6, we conclude the work and highlight future directions.

2 Preliminaries

The advancements in neural networks for tasks like pattern recognition, image classification, and predictive analytics have made them valuable tools for researchers in tackling complex issues. They excel in uncovering hidden patterns in data that can be plotted on a plane but struggle with non-euclidean data [9]. This has led to the development of Geometric Deep Learning to better handle such data. Graphs, which are non-euclidean representations of entities and their relationships, can effectively show connections between two or more elements. They are versatile, as they can be used to model anything, and are particularly useful for modelling non-euclidean network data, which contains a wealth of information. To fully harness its potential, it must be represented formally.

2.1 Graph Neural Network

A simple graph G consists of a non-empty finite set $V(G)$ of elements called nodes (or vertices), and a finite set $E(G)$ of distinct unordered pairs of distinct elements of $V(G)$ called edges. Mathematically, a graph G is defined by Eq. 1 as:

$$G = VE, \tag{1}$$

where V is the vertex set and E is the edge set. For example, a visual representation of a graph G, with vertex set $V(G) = \{a, b, c, d\}$ and edge set $E(G) = \{ab, ac, bc, cd\}$ is shown in Fig. 1.

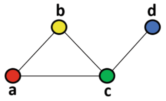


Fig. 1. An example graph.

In Fig. 1, we assume every vertex possesses a feature that defines itself called vertex or node attributes. Graph neural network (GNN) falls under the branch of geometric deep learning, a type of deep learning for emerging methods aiming to generalise deep learning models to non-euclidean domains mainly graphs and manifolds [9]. Early GNN methods were generated with the main goal of constructing a generalisation of CNN architecture on non-euclidean domains. In neural networks, activations are received from preceding layers in order to propagate features to succeeding layers. This is defined in Eq. 2 as:

$$H^{[l+1]} = \sigma(W^{[l]}H^{[l]} + b^{[l]}), \tag{2}$$

where $H^{[l+1]}$ is the feature representation at layer $[l+1]$, σ is a non-linear activation function, $W^{[l]}$ is the weight at the l^{th} layer, $H^{[l]}$ is the feature representation at the l^{th} layer and $b^{[l]}$ is the bias at the l^{th} layer. Kipf et al. [10] defined a Graph Convolution Network (GCN) propagation rule by taking into consideration the adjacency matrix. The adjacency matrix in graph theory shows how nodes are connected to each other. This helps nodes learn information about their neighbours during forward propagation. The adjacency matrix forward propagation can be defined in Eq. 3 as (we eliminate the bias b for simplicity):

$$H^{[l+1]} = \sigma(W^{[l]}H^{[l]}A^*). \tag{3}$$

$$H^{[norm]} = D^{-1}AH. \tag{4}$$

$$H^{[norm]} = D^{-1}\tilde{A}H. \tag{4a}$$

$$H^{[norm]} = D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}H. \tag{4b}$$

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^lW^l). \tag{5}$$

The A^* in Eq. 3 represents the normalised adjacency matrix, normalisation prevents numerical instabilities like vanishing and exploding gradients. Matrix normalisation is the dot product of the inverse of the degree matrix, D^{-1} with the adjacency matrix, A and the node features H as shown in Eq. 4. By implementing a self-loop, a node takes into account its own feature during forward propagation. Setting the diagonal elements of an adjacency matrix to 1 implements self-loops, $\tilde{A}$ represents the refined adjacency matrix in Eq. 4a. Due to varying node degrees, higher degrees nodes tend to dominate. Kipf et al. [10] suggest symmetric normalisation to reduce this dominance in Eq. 4b. After normalizing A with its degree matrix and enforcing self-loop by adding the identity matrix. We get the final equation for forward propagation in GCN as Eq. 5, σ represents a nonlinear activation function like ReLU or Tanh used during neural network computation.

GNN's evaluation and analysis have been more empirical than theoretical, making the entire process seem like a black box. Morris et al. [21] tried to give clarity to this issue by relating GNN operations to the Weisfeiler-Leman graph isomorphism heuristic (1-WL). They show that GNNs possess the same expressiveness as the 1-WL in terms of distinguishing non-isomorphic graphs. Meaning that both algorithms have the same imperfections. They propose a higher-order generalisation of GNNs called k-GNNs which can capture structural information not visible at the node level. In k-GNNs, messaging passing is between sub-graph structures rather than just nodes. Let $V(G)^a$ be a sub-graph of $V(G)$ and b be a set of nodes in $V(G)^a$. The neighbourhood of b is defined below.

$$N(b) = \{t \subset [V(G)]^a \text{ s.t. } |b \cap t \,|= a-1\}\,, \tag{6}$$

Basically, the local neighbourhood $N_L(b)$ consists of all elements in the set $t \subset N(b)$ such that $(v, w) \in E(G)$ for unique $v \in b \setminus t$ and unique $w \in t \setminus b$. The local propagation formula of feature vectors for layer $\ell > 0$ becomes:

$$f^{(\ell)}_{a,\ \mathrm{L}}(s) = \sigma\left(f^{(\ell-1)}_{a,\ \mathrm{L}}(s) \cdot W^{(t)}_1 + \sum_{u \in N_L(s)} f^{(\ell-1)}_{a,\ \mathrm{L}}(u) \cdot W^{(\ell)}_2\right), \tag{7}$$

where $f^{(\ell-1)}_{a,\ \mathrm{L}}(s)$ represents the feature vector of the set of nodes s at layer $t-1$. The aggregation of the local neighbourhood set of s represented by u is $[\sum_{u \in N_L(s)} f^{(\ell-1)}_{a,\ \mathrm{L}}(u)]$. For simplicity, Eq. 7 is denoted as:

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j, \tag{8}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are trainable weight parameters of the GNN, $\mathbf{X}_i$ signifies the feature vectors of set of nodes i. $\sum_{j \in \mathcal{N}(i)}$ denotes the aggregation of the local neighbourhoods of i denoted by j. The edge weight of the local neighbour is denoted by $e_{j,i}$ and $\mathbf{x}_j$ signifies the feature vector of the neighbours. A non-linear function such as ReLU is utilised during computation.

2.2 Pooling

Pooling is popular for dimensionality reduction in convolutional-based systems. In this process, the dimension of the feature map is reduced while retaining useful information and eliminating irrelevant information from the input data. Pooling reduces the complexity of upper layers and simplifies computation by reducing the weight parameters. It also controls over-fitting to a reasonable extent.

Global Mean Pooling. This method of pooling was introduced by Yann LeCun [11]. It returns batch-wise graph-level-outputs by averaging node features across the node dimension. The output for a single graph G_i is computed by:

$$\mathbf{r}_i = \frac{1}{N_i} \sum_{n=1}^{N_i} \mathbf{x}_n \tag{9}$$

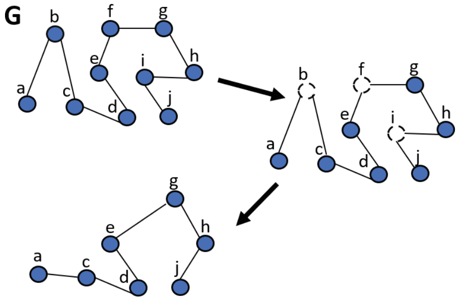


Fig. 2. Condensed graph after top-K pooling. Nodes are dropped after every pooling layer making neighbourhood computation for generating graph embedding easier.

Global Max Pooling. The max pooling [12] method simply passes to the next layer the maximum value within a group of R activations. It returns batch-wise graph-level-outputs by taking the channel-wise maximum across the node dimension. The output for a single graph G_i is computed by:

$$\mathbf{r}_i = \max_{n=1}^{N_i} \mathbf{x}_n \tag{10}$$

For Eqs. 9 and 10, $\mathbf{x}$ represents the node feature matrix.

Top-k Pooling. This is a sort-based method for dimensionality reduction. A projection vector [13] is used to score nodes and only nodes with TopK scores along with related edges are retained. This is an important part of our model design as we perform TopK pooling at every layer to get the optimum embedding for network graph classification. Assuming we have a graph G with vertex set $V(G) = \{a, b, c, d, e, f, g, h, i, j\}$ and edge set $E(G) = \{ab, bc, cd, de, ef, fg, gh, hi, ij\}$, we also assume G is undirected. A visual representation of G is shown in Fig. 2. Suppose we want to train a NN model to get optimum embedding that best defines graphs similar to G. Applying top-K pooling will cause our graph to drop some nodes, as demonstrated in Fig. 2.

It is crucial to note that the nodes carried forward are mathematically and not arbitrarily determined. A measure of how much information is retained after node feature vectors $\mathbf{x}_i$ are projected in the direction of the vector p determines which nodes are dropped and preserved.

3 Proposed Model

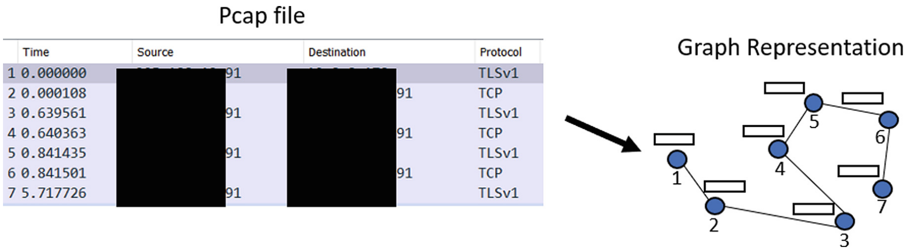
In this section, we first describe the process used in generating our network traffic graphs then define the model used for classification.

3.1 Data Processing

Traffic sessions show bidirectional flows of communication between two or more parties; this makes it a better way of describing packet relationships for classification. Wang et al. demonstrated this [4]. Network communications are processed as packet capture files (pcap or pcapng) for analysis. After collecting packet capture files, we split them into sessions with an open-source tool called splitCap. Packet capture files with *pcapng* extension are converted to *".pcap"* files before splitting occurs. An open-source tool "LibCap" is used to achieve this. The next phase deals with the removal of unwanted information. Network traffic data is collected at the data-link layer that carries information about their physical interfaces. Information contained in the Ethernet header is stripped off as it is not useful for classification and can be spoofed by attackers. Network or flow data like IP addresses and port numbers are not used as features. This restriction forces the model to utilise only the encrypted traffic for classification. At the network layer, the source and destination IP addresses of every packet are masked as they can influence the learning process of the model and can also be easily manipulated by bad actors. At the transport layer, TCP and UDP are used. Since these protocols have different connection orientations with different header sizes (TCP = 20 and UDP = 8), we pad UDP headers with zeros to match TCP. Since our main aim is to classify encrypted traffic we limit our features to information that can't be easily manipulated. Pcap files are then converted to their raw byte format. The maximum transmission unit (MTU) of a packet is 1500 bytes, to maximize information, we extract all packet info and convert it to raw bytes. Where a packet is not up to the MTU, padding is applied. By doing this, every packet has a length of 1500. Next, we normalise every byte to fall within the range of 0–1 by converting every byte to decimal and dividing by 255. Although we conduct tests with different MTU and session sizes, this initial data processing method is the basis for our experiment.

3.2 Model Description

Graph Data Extraction and Creation. The next phase deals with defining and extracting graph information from the pre-processed files. Since network traffic is collected over time, it can be represented as time series information. Traffic sessions are also represented as a back-and-forth exchange over time. Pang et al. [14] generated chained graph structures to represent traffic sessions. Each packet in a session is modelled as a node or vertex, and the edge shows the chronological (time) relationship between packets. Peng et al. [20] conducted experiments to find the number of packets most effective for traffic identification. Their experiment [20] showed the first five-to-seven packets are optimal for classification. We extend this number to ten and extract the first ten packets per session, we choose ten to ensure sufficient exchange of encrypted application data after the completion of the TLS handshake process which is usually three round trips making six packets in Wireshark. A session with ten packets will be modelled as a graph with ten nodes, the edges are bidirectionally connected



	Time	Source	Destination	Protocol
1	0.000000	[redacted]	[redacted]	TLSv1
2	0.000108	[redacted]	[redacted]	TCP
3	0.639561	[redacted]	[redacted]	TLSv1
4	0.640363	[redacted]	[redacted]	TCP
5	0.841435	[redacted]	[redacted]	TLSv1
6	0.841501	[redacted]	[redacted]	TCP
7	5.717726	[redacted]	[redacted]	TLSv1

Fig. 3. Graph representation of traffic session

to their adjacent preceding and succeeding nodes. The entire session is labelled with the application traffic it conveys. Figure 3 demonstrates the structure of our traffic session graph; our network traffic can be represented mathematically as a path graph. To create network graphs, we extract the following information from the pre-processed pcap files. The extracted information is classified into four files, each file consisting of mappings as follows:

Node attributes file: Node information is extracted from the packet, and every node is mapped to its attribute which has an initial feature length of 1500. Nodes represent packets and attributes are packet contents (raw byte).

Edge file: The edge information shows the relationship between nodes in a graph. In the edge file, source nodes are mapped to destination nodes. This preserves the chronological relationship of traffic sessions.

Graph to Label file: After assigning labels to sessions, we extract this mapping of sessions (graphs) and their corresponding class label.

Node to Graph file: As packets belong to particular sessions, we extract this mapping of packets (nodes) and sessions (graphs).

The files are used to create labelled graphs, every graph has nodes with attributes (raw byte) and time series relationships represented by edges.

Architecture. Our architecture as described in Fig. 4, consists of two major parts, the learner and the categorizer. The learner consists of five sub-layers, each sub-layer comprising of a GraphConv or $k-$GNN layer as described by Morris et al., in [21], not be confused with the Graph Convolution layer defined by Kipf et al., in [10]. For simplicity, we refer to the $k-$GNN as GraphConv for the rest of this literature. The next layer is a batch normalisation layer followed by a top-K pooling layer. The aim of the learner is to produce optimum embedding used for classification.

GraphConv [21] is based on a localised higher-order approximation of the Weisfeiler-Leman graph isomorphism heuristic with the propagation rule defined in Eq. 8. This is in contrast to the basic convolutional filter which allows weight sharing (by means of a filter with a fixed kernel size sliding over an input). Convolution in spatial domains recognises similar features irrespective of their

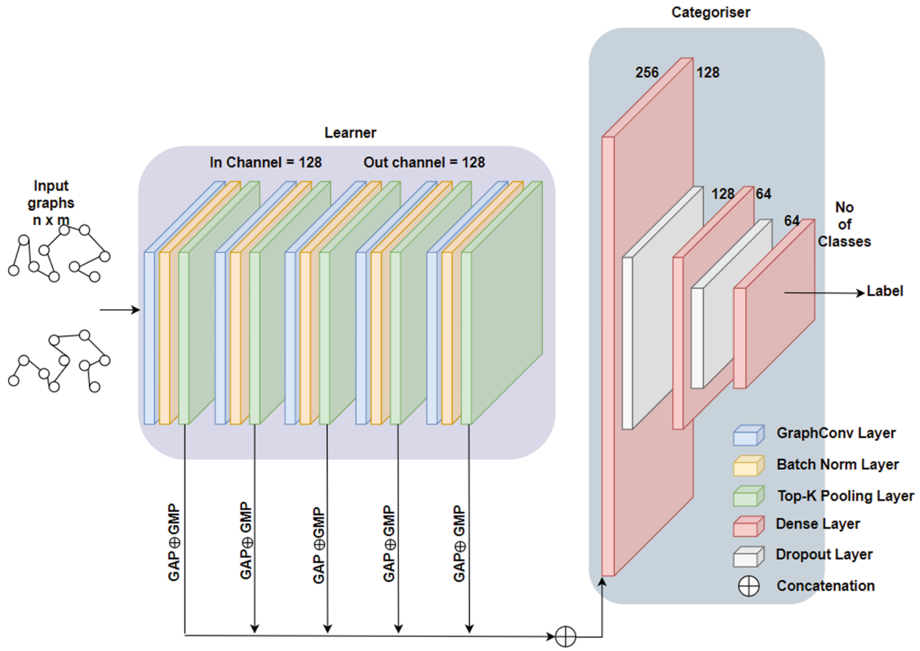


Fig. 4. Proposed encrypted network traffic classifier. Our model consists of two major parts that work independently. The learner, which focuses on generating graph embeddings, and the categorizer which classifies the embedding to its label. The learner consists of five sub-layers, each sub-layer with three internal layers (a GraphConv, batch normalisation and top-k polling layer). The graphconv layer leverages sub-graphs rather than nodes information to perform computation, thereby utilising higher order details of graphs capable of preserving structural info. The number 128 above the internal layers signifies its input and output channel size. The input and output channel sizes remain the same for all sub-layers to deal with the issue of information loss. It is important to note that the input graph reduces in size after every sub-layer, the top-K pooling layer makes this possible. Every sub-layer produces an output which is the concatenated form of its global mean pool and global max pool values, $GMP \oplus GAP$. The five sub-layers produce five embeddings which are concatenated and fed to the categorizer. The symbol $\oplus$ between the learner and categorizer signifies the concatenation function. The number behind and in front of the categorizer's dense layer signifies its input and output channel size. The categorizer takes the concatenated embedding from the learner, passes it through a series of dense layers and finally classifies it. The final layer is a softmax classifier which assigns a label to the input graph.

spatial location. Graphs on the other hand do not have a defined spatial concept. Hence, Kipf et al. [10] utilises spectral graph convolution to lay the mathematical foundation for propagation. Although graph convolution works optimally for node classification, we utilise its potency for the entire graph classification. The model accepts graphs of dimensions $n \times m$, the first GraphConv layer maps the input feature to an output channel of 128 neurons. The selection of this neuron size is based on ease of computation and experiments carried out during this work, literature like [14,15] also use similar sizes. Every GraphConv layer contains 128 neurons. We keep this number constant for all sub-layers in our learner because we extract optimal graph embedding after every sub-layer, since we implement another layer for dimensionality reduction, generating proper embedding becomes key. During our experiments, we realise that a reduction in feature length negatively impacts classification accuracy, a phenomena that occurs in all conducted experiments. This shows that constant reduction of the input and output channel during the learning stage will also impact accuracy. To deal with this issue of loss of information we keep the channel sizes constant and implement dimensionality reduction with a top-K pooling layer. This way, node features defining the graphs are well represented before pooling. The non-linear function used is the Rectified Linear Unit (ReLU).

Following the GraphConv is the batch normalisation (BN) layer. Normalisation is an important aspect of our whole process and it is implemented at the data processing and model training level. We introduce this layer to deal with the internal covariate shift. BN layers standardise the mean and variance of each unit in order to stabilise learning, making the gradients more predictive and increasing convergence time. We utilise BN to achieve a stable distribution of activation values during the training process, hence, it is introduced after the non-linearity. BN layers have also been proven to reduce over-fitting in convolutional NN. Garbin et al. [8] in an experiment found BN layers a better choice to improve accuracy and advised BN layers to be considered as an initial means to deal with over-fitting before other methods.

The next layer is the top-K pooling layer, as described in Sect. 2.2 the aim of this layer is overall dimensionality reduction of graphs. We employ this method because of the irregular sizes of network packets per session, during data processing we noticed some sessions with over 50 packets. Similar to pooling in CNN, top-K pooling reduces the number of nodes in the input graph every time it is used. The nature of our graphs demands a way to optimise feature propagation for ease of computation. Hence, we apply a top-K pooling layer to make our model robust. We still perform more experiments with the truncated network sessions for impact analysis. The Top-K pooling layer uses a trainable projection vector [13] to select top-K nodes for the next layer. Dimensionality reduction is also implemented to deal with over-smoothness in our model. Over-smoothing is a phenomenon where GNN performance gradually reduces with increasing layers, this occurs when node embedding becomes similar the GNN does not seem to learn anything new. Sadowski et al. [16] demonstrated how dimensionality reduction alleviates this issue.

To get embeddings for graph classification, we compute the global mean pooling and global max pooling after every sub-layer. This embedding is concatenated for every sub-layer of the learner, we end up with five embeddings, one per sub-layer of our learner. The five embeddings are concatenated and fed to the categorizer.

The categorizer consists of two sub-layers and a softmax classifier. The categorizer can be seen as a simple feed-forward NN with three linear or dense layers. The concatenated embeddings from the learner are fed to the categorizer which then predicts the label of the graph based on learned parameters. Since the output of the learner is of dimension 2 × 128, the first sub-layer of the categorizer has an input channel of 256 and an output channel of 128. We introduce a dropout layer with a probability of 0.5 to tackle over-fitting. The second sub-layer has an input channel of 128 and an output channel of 64, we also implement a dropout layer with the same probability here. The final layer has an input channel of 64 and an output channel equal to the total number of labels. Each with a probability that defines how well a graph fits the label, this is achieved with a softmax classifier.

4 Evaluation

To validate the robustness of our model, we subject it to a series of tests with a range of datasets. The evaluation process is divided into four sections as follows. The dataset, here we give an in-depth description of the dataset and why we choose it for evaluation. In the experimental methodology section, we explicitly define the processes and types of experiments conducted, the parameters with respect to the model training and testing phase are also defined. In Sect. 4.3 we discuss the results of our experiments and finally compare our results with state-of-the-art.

4.1 Datasets

The VPN-nonVPN dataset (ISCXVPN2016) [17] captures real traffic of users Alice and Bob created to use services (applications) described in Table 1. Seven traffic classes are captured for VPN and non-VPN (at the time of dataset collection the P2P class of the non-VPN traffic was removed from the repository). The VPN-nonVPN dataset [17] is one of the most popular datasets for encrypted traffic classification tasks, making it suitable for comparative analysis. Reviewed

Table 1. VPN-nonVPN Dataset Summary

Traffic	Content
Web Browsing	Firefox and Chrome
Email	SMPTS, POP3S and IMAPS
Chat	ICQ, AIM, Skype, Gmailchat, Facebook and Hangouts
Streaming	Vimeo, Youtube and Spotify
File Transfer	Skype, FTPS and SFTP
VoIP	Facebook, Skype and Hangouts voice calls
P2P	Vimeo, Youtube and Spotify

Table 2. USTC-TFC2016 Dataset Summary

Traffic	Content
Benign	BitTorrent, Facetime, FTP, Gmail, MySQL, Outlook, Skype, SMB, Weibo, WorldOfWarcraft
Malware	Cridex, Geodo, Htbot, Miuref, Neris, Nsis-ay, Shifu, Tinba, Virut, Zeus

literature like [4,15] utilised this dataset for their analysis. The dataset is in *.pcap* and *.pcapng* format and 28 GB in size.

The USTC-TFC2016 dataset [18] consists of two parts, the malware class and benign class. The malware class consists of ten types of malware traffic collected from public websites in a real network environment. The benign class consists of ten types of normal traffic collected using a professional traffic simulation equipment. This dataset is popular for traffic classification tasks and suitable for comparative analysis. The dataset is in pcap format and 3.71GB in size. Table 2 gives a summary of the USTC-TFC2016 dataset.

4.2 Experimental Methodology

To explicitly demonstrate our contribution and prove the robustness of our model, we conduct experiments using the datasets defined in Sect. 4.1. The VPN-nonVPN dataset is split so separate experiments are carried out on the *VPN traffic* and *NonVPN traffic*. For the USTC-TFC dataset, separate experiments are conducted on the *benign* and *malware* traffic respectively. Network graphs are created from traffic sessions. The traffic session distribution for the VPN non-VPN dataset is described in Table 3. Real-world datasets are naturally imbalanced, and classifiers have to incorporate augmentation techniques during training. The USTC-TFC dataset has an abundance of samples, so we extracted 2000 sessions per application.

The different input parameter sizes for our experiments are defined in Table 4. For every input parameter, we process the datasets as required and conduct experiments. For the first experiment, we truncate at the sessions layer, making every session and resulting graph have a fixed number of packets (nodes). For the second experiment, truncation isn't applied, the number of nodes per graph varies as packets per session vary. For experiment three, we truncate the datalink and session layer. We slash the packet size by almost half the MTU, ending

Table 3. VPN-nonVPN dataset sessions sample distribution

Class	Chat	Email	File	P2P	Stream	VoIP
VPN	4029	298	1020	477	659	11985
non-VPN	6523	7312	276	–	445	1781

Table 4. Input Graph Specifications

Experiment	Nodes per graph	Node attribute size
1	10 nodes (fixed)	1500
2	Variable (not fixed)	1500
3	10 nodes (fixed)	784
4	Variable (not fixed)	784

up with 784 bytes per packet and also keeping the sessions fixed at 10 packets per session. The choice of 784 is motivated by literature that utilise images from network traffic [6,19] to address the classification problem. Zou et al. [6] proved that network images of size 784 bytes are effective for traffic classification, and are more lightweight than 1500 bytes. Wang et al. [19] generated network traffic images and used only the first 784 bytes to get fixed sized input for their CNN model. For experiment four, truncation is applied only at the data-link layer with packet sizes slashed down to 784 bytes.The first experiment lays the foundation for comparative analysis while demonstrating the generalisation strength of our model. Other experiments are conducted for impact analysis of variable length input (data truncation) on encrypted network traffic classification and to demonstrate the model generalisation strength.

Training Specification. The following specifications are used for developing, training and testing. The PyTorch geometric library with a python 3.9.13 backend is used to generate the graphs, build, train and test our model. The hardware specification is a Linux dell 5.15.0-56-generic server, the processor is a 12th Gen Intel(R) Core(TM) i9-12900, 125 GB of physical RAM and a NVIDIA RTX A4000 GPU.

During the training process, weighted random sampling is implemented to deal with the imbalanced nature of the VPN Non-VPN dataset. Weighted random sampling ensures minority classes are properly represented during the training process. For all experiments, we split our datasets into two, 80% for training and 20% for testing. The hyper-parameters of the model are as follows: batch size is 256, the number of epochs is 500 for regular training and 200 during cross validation. The Adam optimizer is used to improve the categorical cross-entropy loss function with a learning rate of 0.0003, and the decay rate is 0.00001. We used the same training specifications for all experiments. We evaluate our model using the four standard classification metrics namely, Precision, Recall, $f1$Score and Accuracy.

Test for Over-Fitting. The increase in parameters during training can cause neural networks to adapt so much to a particular data that it performs poorly when unseen data is introduced. Ensuring a model does not overfit is a crucial aspect of neural network training. To demonstrate the generalisation of our model, we subject it to a cross-validation test. For this test, we perform stratified cross-validation on our model with the dataset. Our choice of stratified cross-validation is motivated by the imbalanced nature of the dataset, and we need to ensure that classes are well represented during training. We use the same training specification as Sect. 4.2 with few modifications. We split the data set into 5 folds and reduce the epoch to 200 to limit the tendency of the model adapting to the training data.

4.3 Results

A total of 12 experiments are conducted with our model on two datasets. Four experiments on the VPN and four on the non-VPN traffic of the ISCXVPN dataset. Two experiments on the benign and two on the malware traffic of the USTC-TFC dataset. To validate our results, we perform stratified cross-validation with a fold of 5 on the dataset.

Table 5 shows the classification of VPN traffic. Our model yields its best result in the first experiment when truncation is applied at the session layer and padding at the data-link layer. The model generalises optimally for the first three experiments but struggles in the last experiment when truncation is applied only at the data-link layer. For experiments 2 and 4 we use graphs with variable vertex cardinality for classification. The results demonstrate that geometric deep learning, like other DL methods, thrives when the input size is fixed. In situations where the input parameter sizes vary, such as in experiment 4, a compensation to improve accuracy should be well-defined node attributes of an adequate size.

Table 6 shows very similar results to Table 5 for the non-VPN traffic classification task. The best result is achieved in the first experiment. Applying

Table 5. VPN Traffic classification result

Input Spec	Metric	Chat	Email	File	Stream	P2P	VoIP	Accuracy
10nodes/1500	Precision	0.9738	0.9672	0.9069	0.8993	0.9877	0.9850	0.9784
	Recall	0.9595	0.9363	0.8685	0.9921	0.9877	0.9891	
	$F1$score	0.9666	0.9516	0.8873	0.9434	0.9877	0.9871	
v-nodes/1500	Precision	0.9831	0.8983	0.6916	0.7484	0.9444	0.9867	0.9437
	Recall	0.9509	0.8413	0.9181	0.8406	0.9533	0.9512	
	$F1$score	0.9409	0.8689	0.7889	0.7918	0.9488	0.9687	
10nodes/784	Precision	0.9674	0.8955	0.7103	0.8529	0.9700	0.9838	0.9548
	Recall	0.9601	0.9231	0.8861	0.8722	0.9417	0.9648	
	$F1$score	0.9637	0.9091	0.7885	0.8625	0.9557	0.9742	
v-nodes/784	Precision	0.2273	0.1908	0.2559	0.5215	0.3333	0.7080	0.4312
	Recall	0.4065	0.3906	0.4645	0.6693	0.6790	0.4168	
	$F1$score	0.2916	0.2564	0.3300	0.5862	0.4472	0.5247	

Table 6. non-VPN Traffic classification result

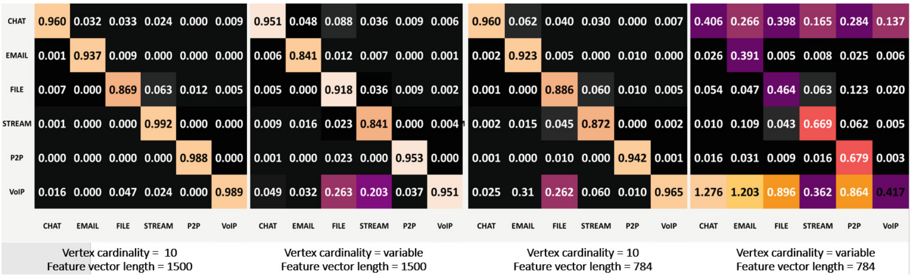
Input Spec	Metric	Chat	Email	File	Stream	VoIP	Accuracy
10nodes/1500	Precision	0.8683	0.8389	0.9412	1.000	1.000	0.8707
	Recall	0.8139	0.8881	0.9412	0.9867	0.9745	
	$F1$score	0.8402	0.8586	0.9412	0.9933	0.9871	
v-nodes/1500	Precision	0.8279	0.6483	0.6667	0.8493	0.8697	0.7160
	Recall	0.4345	0.9241	0.7568	0.8732	0.8566	
	$F1$score	0.5699	0.7620	0.7089	0.8611	0.8631	
10nodes/784	Precision	0.6823	0.5266	0.8182	0.7719	0.8392	0.5859
	Recall	0.2465	0.9023	0.7660	0.7333	0.5604	
	$F1$score	0.3622	0.6650	0.7912	0.7521	0.6720	
v-nodes/784	Precision	0.5852	0.5516	0.2400	0.4017	0.5736	0.5435
	Recall	0.2469	0.7662	0.6000	0.6026	0.6632	
	$F1$score	0.3473	0.6414	0.3429	0.4821	0.6151	

truncation at the data-link layer yields the worst result. Notice how the model struggles to classify the chat and email traffic across all experiments. This is attributed to poor distinction of applications within network sessions. In our work, every session has a label that defines its traffic class. It is possible to have multiple application leveraging the same session or have a case of tunnelling where packets are wrapped inside packets. Our model does not pay attention to this dynamics rather it focuses on distinguishing a particular traffic session by labelling it. The imbalanced nature of the traffic also contributes a great deal to this confusion. Figures 5 and 6 show the confusion matrix for the VPN and Non-VPN traffic of the ISCXVPN dataset.

Figure 7 shows the confusion matrix for the benign class of the USTC-TFC dataset. With a balanced dataset, our model perfectly classifies six applications for the first experiment and classifies five applications perfectly when the feature vector is slashed by almost half. As demonstrated in Fig. 8 our model shows powerful generalisation strength for the malware class of the USTC-TFC dataset. We get very similar results for both experiments regardless of data truncation. This time we test to see the best feature length for optimum generalisation. Hence, we keep the nodes length fixed at 10 while varying the attribute size. Our model maintains an accuracy of over 90% after we slash the feature vector by almost half. For other evaluation metrics, our model again maintains accuracies of over 90%. The experiments proved that an increase in features size while keeping vertex cardinality fixed, improves generalisation strength. Table 7 shows the overall performance of our model after all 12 experiments are carried out.

4.4 Performance Comparison

We compare our best results with state-of-the-art results that use the same dataset and a similar data processing approach. For VPN classification, our



Vertex cardinality = 10
Feature vector length = 1500

	CHAT	EMAIL	FILE	STREAM	P2P	VoIP
CHAT	0.960	0.032	0.033	0.024	0.000	0.009
EMAIL	0.001	0.937	0.009	0.000	0.000	0.000
FILE	0.007	0.000	0.869	0.063	0.012	0.005
STREAM	0.001	0.000	0.000	0.992	0.000	0.000
P2P	0.000	0.000	0.000	0.000	0.988	0.000
VoIP	0.016	0.000	0.047	0.024	0.000	0.989

Vertex cardinality = variable
Feature vector length = 1500

	CHAT	EMAIL	FILE	STREAM	P2P	VoIP
CHAT	0.951	0.048	0.088	0.036	0.009	0.006
EMAIL	0.006	0.841	0.012	0.007	0.000	0.001
FILE	0.005	0.000	0.918	0.036	0.009	0.002
STREAM	0.009	0.016	0.023	0.841	0.000	0.004
P2P	0.001	0.000	0.023	0.000	0.953	0.000
VoIP	0.049	0.032	0.263	0.203	0.037	0.951

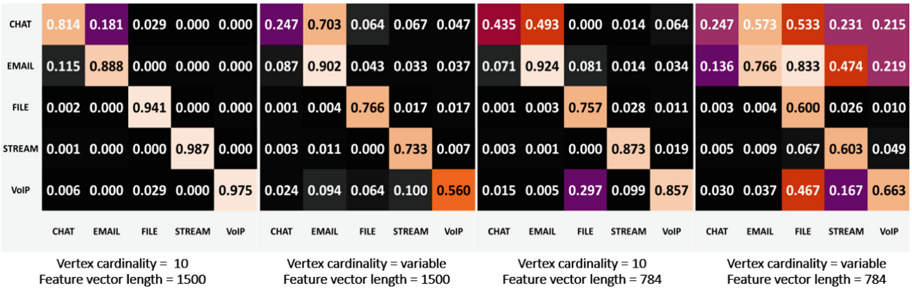
Vertex cardinality = 10
Feature vector length = 784

	CHAT	EMAIL	FILE	STREAM	P2P	VoIP
CHAT	0.960	0.062	0.040	0.030	0.000	0.007
EMAIL	0.002	0.923	0.005	0.000	0.010	0.000
FILE	0.001	0.000	0.886	0.060	0.010	0.005
STREAM	0.002	0.015	0.045	0.872	0.000	0.002
P2P	0.001	0.000	0.010	0.000	0.942	0.001
VoIP	0.025	0.31	0.262	0.060	0.010	0.965

Vertex cardinality = variable
Feature vector length = 784

	CHAT	EMAIL	FILE	STREAM	P2P	VoIP
CHAT	0.406	0.266	0.398	0.165	0.284	0.137
EMAIL	0.026	0.391	0.005	0.008	0.025	0.006
FILE	0.054	0.047	0.464	0.063	0.123	0.020
STREAM	0.010	0.109	0.043	0.669	0.062	0.005
P2P	0.016	0.031	0.009	0.016	0.679	0.003
VoIP	1.276	1.203	0.896	0.362	0.864	0.417

Fig. 5. Confusion matrix for VPN dataset



Vertex cardinality = 10
Feature vector length = 1500

	CHAT	EMAIL	FILE	STREAM	VoIP
CHAT	0.814	0.181	0.029	0.000	0.000
EMAIL	0.115	0.888	0.000	0.000	0.000
FILE	0.002	0.000	0.941	0.000	0.000
STREAM	0.001	0.000	0.000	0.987	0.000
VoIP	0.006	0.000	0.029	0.000	0.975

Vertex cardinality = variable
Feature vector length = 1500

	CHAT	EMAIL	FILE	STREAM	VoIP
CHAT	0.247	0.703	0.064	0.067	0.047
EMAIL	0.087	0.902	0.043	0.033	0.037
FILE	0.001	0.004	0.766	0.017	0.017
STREAM	0.003	0.011	0.000	0.733	0.007
VoIP	0.024	0.094	0.064	0.100	0.560

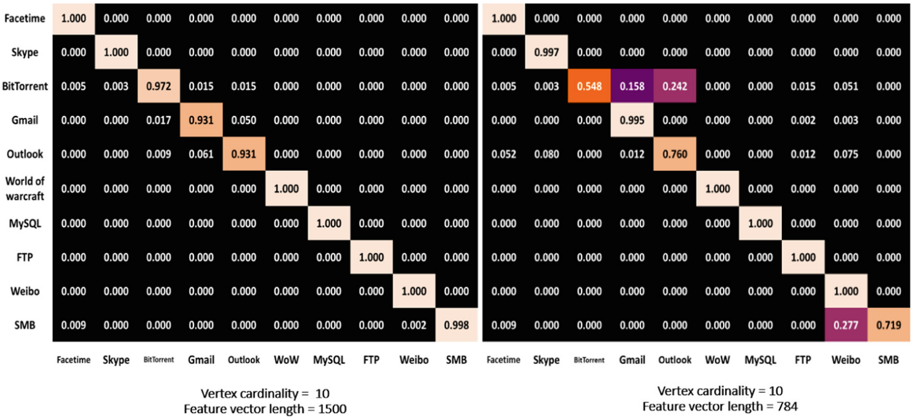
Vertex cardinality = 10
Feature vector length = 784

	CHAT	EMAIL	FILE	STREAM	VoIP
CHAT	0.435	0.493	0.000	0.014	0.064
EMAIL	0.071	0.924	0.081	0.014	0.034
FILE	0.001	0.003	0.757	0.028	0.011
STREAM	0.003	0.001	0.000	0.873	0.019
VoIP	0.015	0.005	0.297	0.099	0.857

Vertex cardinality = variable
Feature vector length = 784

	CHAT	EMAIL	FILE	STREAM	VoIP
CHAT	0.247	0.573	0.533	0.231	0.215
EMAIL	0.136	0.766	0.833	0.474	0.219
FILE	0.003	0.004	0.600	0.026	0.010
STREAM	0.005	0.009	0.067	0.603	0.049
VoIP	0.030	0.037	0.467	0.167	0.663

Fig. 6. Confusion matrix for non-VPN dataset



Vertex cardinality = 10
Feature vector length = 1500

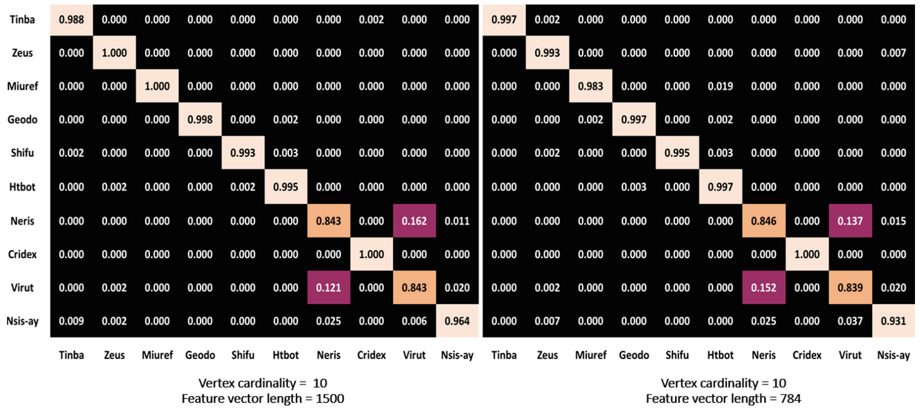
	Facetime	Skype	BitTorrent	Gmail	Outlook	WoW	MySQL	FTP	Weibo	SMB
Facetime	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Skype	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BitTorrent	0.005	0.003	0.972	0.015	0.015	0.000	0.000	0.000	0.000	0.000
Gmail	0.000	0.000	0.017	0.931	0.050	0.000	0.000	0.000	0.000	0.000
Outlook	0.000	0.000	0.009	0.061	0.931	0.000	0.000	0.000	0.000	0.000
World of warcraft	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
MySQL	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
FTP	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
Weibo	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
SMB	0.009	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.998

Vertex cardinality = 10
Feature vector length = 784

	Facetime	Skype	BitTorrent	Gmail	Outlook	WoW	MySQL	FTP	Weibo	SMB
Facetime	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Skype	0.000	0.997	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BitTorrent	0.005	0.003	0.548	0.158	0.242	0.000	0.000	0.015	0.051	0.000
Gmail	0.000	0.000	0.000	0.995	0.000	0.000	0.000	0.002	0.003	0.000
Outlook	0.052	0.080	0.000	0.012	0.760	0.000	0.000	0.012	0.075	0.000
World of warcraft	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
MySQL	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
FTP	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
Weibo	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
SMB	0.009	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.277	0.719

Fig. 7. Confusion matrix for USTC-TFC Benign Traffic

model generalises optimally with approximately the same value across all evaluation metrics. Our model maintains an accuracy that doesn't fall below 97%, an occurrence not demonstrated by state-of-the-art models. The results of compared literature remained unstable for different evaluation metrics (Table 8).



	Tinba	Zeus	Miuref	Geodo	Shifu	Htbot	Neris	Cridex	Virut	Nsis-ay
Tinba	0.988	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.000	0.000
Zeus	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Miuref	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Geodo	0.000	0.000	0.000	0.998	0.000	0.002	0.000	0.000	0.000	0.000
Shifu	0.002	0.000	0.000	0.000	0.993	0.003	0.000	0.000	0.000	0.000
Htbot	0.000	0.002	0.000	0.000	0.002	0.995	0.000	0.000	0.000	0.000
Neris	0.000	0.000	0.000	0.000	0.000	0.000	0.843	0.000	0.162	0.011
Cridex	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
Virut	0.000	0.002	0.000	0.000	0.000	0.000	0.121	0.000	0.843	0.020
Nsis-ay	0.009	0.002	0.000	0.000	0.000	0.000	0.025	0.000	0.006	0.964

Vertex cardinality = 10
Feature vector length = 1500

	Tinba	Zeus	Miuref	Geodo	Shifu	Htbot	Neris	Cridex	Virut	Nsis-ay
Tinba	0.997	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Zeus	0.000	0.993	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.007
Miuref	0.000	0.000	0.983	0.000	0.000	0.019	0.000	0.000	0.000	0.000
Geodo	0.000	0.000	0.002	0.997	0.000	0.002	0.000	0.000	0.000	0.000
Shifu	0.000	0.002	0.000	0.000	0.995	0.003	0.000	0.000	0.000	0.000
Htbot	0.000	0.000	0.000	0.003	0.000	0.997	0.000	0.000	0.000	0.000
Neris	0.000	0.000	0.000	0.000	0.000	0.000	0.846	0.000	0.137	0.015
Cridex	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
Virut	0.000	0.002	0.000	0.000	0.000	0.000	0.152	0.000	0.839	0.020
Nsis-ay	0.000	0.007	0.000	0.000	0.000	0.000	0.025	0.000	0.037	0.931

Vertex cardinality = 10
Feature vector length = 784

Fig. 8. Confusion matrix for USTC-TFC Malware Traffic

Table 7. Overall Performance Summary

Dataset	Input Spec	Precision	Recall	$F1$Score	Accuracy
VPN	10nodes/1500	0.9749	0.9748	0.9747	0.9748
	10nodes/784	0.9586	0.9548	0.9561	0.9548
	v-nodes/1500	0.9494	0.9437	0.9455	0.9437
	v-nodes/784	0.5543	0.4312	0.4587	0.4312
non-VPN	10nodes/1500	0.8722	0.8707	0.8706	0.8707
	10nodes/784	0.6398	0.5859	0.5466	0.5859
	v-nodes/1500	0.7505	0.7160	0.6978	0.7160
	v-nodes/784	0.5588	0.5434	0.5144	0.5435
Benign	10nodes/1500	0.9830	0.9831	0.9831	0.9830
	10nodes/784	0.9177	0.9030	0.8983	0.9030
Malware	10nodes/1500	0.9655	0.9655	0.9655	0.9655
	10nodes/784	0.6578	0.9573	0.9575	0.9573

For the task of non-VPN classification, we obtain similar results to the compared literature. Our model still demonstrates optimum generalisation strength producing accuracies that do not fall below 87%. Gil et al. [17] using flow-based features got a precision score of 90.6% on the non-VPN dataset. While this result is impressive, applications share similar flow-based features which can impact classification. Huoh et al. [15] GNN-based classifier performed well for both tasks; their model does not generalise optimally for VPN classification. Comparatively, Song et al. [23] achieved the closest accuracy to our work; they used text-based CNN for classification. This closeness in results can be attributed to "sense of locality". GNNs and CNNs utilise locality relationships for the classification tasks. While GNNs can be built with higher order capability to perform

Table 8. Performance Comparison

Dataset	Metric	DeepPacket (GAE) [5]	Gil et al. [17]	Song et al. [23]	1DCNN [4]	Huoh et al. [15]	k-NN. [24]	Our work
VPN	Precision	0.97	0.89	95.2	0.949	0.913	–	**0.9748**
	Recall	0.80	–	97.2	0.973	0.940	–	**0.9784**
	$F1$Score	0.97	–	96.1	–	0.926	–	**0.9747**
	Accuracy	–	–	–	–	–	0.94	**0.9748**
non-VPN	Precision	0.87	**0.906**	87.6	0.855	0.882	–	0.8722
	Recall	**0.88**	–	87.3	0.858	0.866	–	0.8707
	$F1$Score	0.87	–	87.5	–	0.871	–	**0.8706**
	Accuracy	–	–		–	–		0.8707

better, results from both methods should not be far apart. Most state-of-the-art models avoid the use of accuracy as a metric for evaluation. This is a normal occurrence when the utilised dataset is imbalanced. The accuracy of classifier on imbalanced dataset does not depict the true nature of the model's performance as minority classes can negatively impact accuracy. For our work, we consider all four evaluation metric (including accuracy) as in the real world, dataset is mostly imbalanced.

5 Related Work

Deep learning methods have been immensely applied to the task of traffic classification, recurrent(RNN) and convolutional neural networks (CNN) [7] account for the most usage. In analysing network traffic, the data processing stage is crucial to the overall process. A common practice by state-of-the-art literature [4–6] is utilising raw packet bytes for classification. Extracted byte information is processed to have a fixed length as this is a requirement for most deep learning approaches. Data truncation is applied, and inputs are fed to the network for classification. The model is left with the task of making sense of the truncated input. While reviewed state-of-the-art models provide good accuracy, it is important to note that generalisation isn't guaranteed as information lost during data processing is necessary for classification.

Geometric deep learning is an emerging concept that generalise neural networks to non-euclidean domains. Graphs are a visual way to demonstrate relationships between two or more entities. Since network data exists in non-euclidean space, they can be represented as graphs. Unlike RNN and CNN, GNN can perform computation on variable-sized inputs. Huoh et al. [15] applied GNN to traffic classification, they conduct a range of experiments which produced good results. When they use raw bytes for classification, they experience significant drop in accuracy. Their design does not utilise structural information necessary for graph classification. GNNs rely on message passing to propagate information across nodes, this propagation method performs well for the task of node classification. To classify the entire graph, a stronger propagation function is necessary. Shen et al. [22] used GNN to distinguish decentralised application.

They design simple traffic dispersion graphs representing applications then use GNN for classification. They achieve promising results, attributed to the simplistic nature of their graphs. When network traffic sessions are modelled as variable sized inputs, the tendency of having graphs with huge vertex cardinality increases. In such scenarios, the task of graph classification demands higher order GNNs that generate optimum embedding for graph classification.

A process not demonstrated by the reviewed literature is defining the impact of lost information on a model's performance. The practice of truncating network packet data leads to loss of classification information, discarding information impacts the generalisation strength of classifiers and should be analysed before implemented. Literature that utilise GNN for the task of classification failed to utilise the potential of graphs by taking advantage of structural details for graph classification. The imbalance nature of dataset should always be taken into consideration during training to produce model that generalise optimally.

6 Conclusion

This work presents a higher order GNN for the task of encrypted traffic classification. We build on theoretical foundations that relates the short fall of basic GNN to that of the Weisfeiler-Leman graph isomorphism. We observe that traditional machine learning and deep learning based methods demands fixed sized input for classification. This requirement causes information discarding during data processing leading to ordinary representations of traffic data. As graphs are a powerful way to demonstrate relationships between entities, we model traffic session as graphs. Nodes represents packets and edges define chronological relationship between nodes. We harness this expressive nature of graphs to model network traffic with different input specifications, defining a baseline data processing method that conforms to state-of-the-art. The selected datasets are based on popularity within the research domain, and their suitability for comparative analysis. Our Proposed model which consist of two major parts is capable of preserving vertex and structural information of graphs suitable for classification. In training our model we consider the imbalance nature of network data and define ways to curb its ripple effect. When compared to state-of-the art, our model demonstrates optimum generalisation strength on all dataset across all experiment conducted. We conduct further test to determine the impact of truncation on traffic classification. Based on the evaluation metric from series of test, our model's overall aim was achieved. Higher order graphs proved suitable for the task of encrypted network traffic classification. In improving the current work, the focus needs to shift from the classification stage to feature extraction stage. To harness the full potential of GNN, fined grained graph structures need to be developed, as graphs can be spectral or spatial in nature. A hybrid of both methods for feature representation can aid classification. An extension of the current method to restricted network contexts, such as IoT, IIoT or CPS is important. Finally, unknown traffic classification should be explored.

References

1. SonicWall 2022 SonicWall Cyber Threat Report: Cyberattacks Climb Due to Seismic Shift in Geopolitical Landscape (SonicWall, 2022)
2. Zhang, J., Chen, X., Xiang, Y., Zhou, W., Wu, J.: Robust network traffic classification. IEEE/ACM Trans. Netw. **23**, 1257–1270 (2014)
3. Google Google Transparency Report, HTTPS encryption on the web (2023). https://transparencyreport.google.com/https/overview?hl=en
4. Wang, W., Zhu, M., Wang, J., Zeng, X., Yang, Z.: End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In: IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 43–48 (2017)
5. Lotfollahi, M., Jafari Siavoshani, M., Shirali Hossein Zade, R., Saberian, M.: Deep packet: a novel approach for encrypted traffic classification using deep learning. Soft Comput. **24**, 1999–2012 (2020)
6. Zou, Z., Ge, J., Zheng, H., Wu, Y., Han, C., Yao, Z.: Encrypted traffic classification with a convolutional long short-term memory neural network. In: IEEE 20th International Conference on High Performance Computing And Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference On Data Science And Systems (HPCC/SmartCity/DSS), pp. 329–334 (2018)
7. Okonkwo, Z., Foo, E., Li, Q., Hou, Z.: A CNN based encrypted network traffic classifier. Aust. Comput. Sci. Week **2022**, 74–83 (2022)
8. Garbin, C., Zhu, X., Marques, O.: Dropout vs. batch normalization: an empirical study of their impact to deep learning. Multimed. Tools Appl. **79**, 12777–12815 (2020)
9. Bronstein, M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond Euclidean data. IEEE Signal Process. Mag. **34**, 18–42 (2017)
10. Kipf, T., Welling, M.: Semi-supervised classification with graph convolutional networks. ArXiv Preprint ArXiv:1609.02907 (2016)
11. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**, 2278–2324 (1998)
12. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, pp. 253–256 (2010)
13. Gao, H., Ji, S.: Graph U-nets. In: International Conference on Machine Learning, pp. 2083–2092 (2019)
14. Pang, B., Fu, Y., Ren, S., Wang, Y., Liao, Q., Jia, Y.: CGNN: traffic classification with graph neural network. ArXiv Preprint ArXiv:2110.09726 (2021)
15. Huoh, T., Luo, Y., Li, P., Zhang, T.: Flow-based encrypted network traffic classification with graph neural networks. IEEE Trans. Netw. Serv. Manage. 1–1 (2022). https://doi.org/10.1109/TNSM.2022.3227500
16. Sadowski, K., Szarmach, M., Mattia, E.: Dimensionality reduction meets message passing for graph node embeddings. ArXiv Preprint ArXiv:2202.00408 (2022)
17. Draper-Gil, G., Lashkari, A., Mamun, M., Ghorbani, A.: Characterization of encrypted and VPN traffic using time-related. In: The 2nd International Conference on Information Systems Security and Privacy (ICISSP), pp. 407–414 (2016)
18. CTU-University The Stratosphere IPS Project Dataset (2016). https://stratosphereips.org/category/dataset.html
19. Wang, W., Zhu, M., Zeng, X., Ye, X., Sheng, Y.: Malware traffic classification using convolutional neural network for representation learning. In: 2017 International Conference on Information Networking (ICOIN), pp. 712–717 (2017)

20. Peng, L., Yang, B., Chen, Y., Wu, T.: How many packets are most effective for early stage traffic identification: an experimental study. China Commun. **11**, 183–193 (2014)
21. Morris, C., et al.: Weisfeiler and leman go neural: higher-order graph neural networks. CoRR. abs/1810.02244 (2018). http://arxiv.org/abs/1810.02244
22. Shen, M., Zhang, J., Zhu, L., Xu, K., Du, X.: Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. IEEE Trans. Inf. Forensics Secur. **16**, 2367–2380 (2021)
23. Song, M., Ran, J., Li, S.: Encrypted traffic classification based on text convolution neural networks. In: 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), pp. 432–436 (2019)
24. Yamansavascilar, B., Guvensan, M., Yavuz, A., Karsligil, M.: Application identification via network traffic classification. 2017 International Conference on Computing, Networking and Communications (ICNC), pp. 843–848 (2017)

Author Index

L. Simpson and M. A. Rezazadeh Baee (Eds.): ACISP 2023, LNCS 13915, pp. 651–652, 2023.
https://doi.org/10.1007/978-3-031-35486-1