

Arpita Patra
Nigel P. Smart (Eds.)

LNCS 10698

Progress in Cryptology – INDOCRYPT 2017

18th International Conference on Cryptology in India
Chennai, India, December 10–13, 2017
Proceedings

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, Lancaster, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Zurich, Switzerland

John C. Mitchell

Stanford University, Stanford, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Dortmund, Germany

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbrücken, Germany

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

Arpita Patra · Nigel P. Smart (Eds.)

Progress in Cryptology – INDOCRYPT 2017

18th International Conference on Cryptology in India
Chennai, India, December 10–13, 2017
Proceedings

Editors

Arpita Patra
Indian Institute of Science (IISc)
Bengaluru
India

Nigel P. Smart
Department of Computer Science
University of Bristol
Bristol
UK

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-71666-4 ISBN 978-3-319-71667-1 (eBook)
<https://doi.org/10.1007/978-3-319-71667-1>

Library of Congress Control Number: 2017959624

LNCS Sublibrary: SL4 – Security and Cryptology

© Springer International Publishing AG 2017

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

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

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

Printed on acid-free paper

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

Preface

INDOCRYPT 2017, the 18th International Conference on Cryptology in India, was held at Institute of Mathematical Sciences, Chennai, India, during December 10–13, 2017. The INDOCRYPT series of conferences began in 2000 under the leadership of Prof. Bimal Roy of the Indian Statistical Institute and is organized under the aegis of the Cryptology Research Society of India (CRSI). The conference focused on all technical aspects of cryptology.

The submissions for INDOCRYPT 2017 were due on August 27, 2017. In response to the call for papers, we received 75 submissions from around 20 countries, out of which 19 were chosen for inclusion in the program. The review process was conducted in two stages. In the first stage, each paper was reviewed by at least three independent reviewers, with papers from Program Committee members receiving at least five reviews. This was followed by a week-long rigorous and detailed discussion phase to decide on the acceptance of the submissions. Reviewers with potential conflicts of interest for specific papers were excluded from all discussions about those papers. The 43 members of the Program Committee were aided in this tedious and time-consuming task by many external reviewers. We would like to thank them all for their service, their expert opinions, and their spirited contributions to the review process. The authors had to revise their papers according to the suggestions of the referees and submit the camera-ready versions by October 15.

The submission and review process was done using Shai Halevi’s Web Submission and Review Software. We wish to express our sincere gratitude to Shai Halevi for the software, which facilitated a smooth and easy submission and review process.

INDOCRYPT 2017 had three invited speakers with two from academia and one from the Government of India. Elette Boyle (Israel) enlightened the audience on “Recent Advances in Function and Homomorphic Secret Sharing”. Tançrède Lepoint (USA) spoke on the interesting topic of “Post-Quantum Cryptography Using Module Lattices”. The speech of Saikat Datta (Policy Director, Centre for Internet & Society, India) covered policy-making in India on Cryptography.

Finally, we would like to thank the general chairs, Prof. C. Pandu Rangan (Indian Institute of Technology Madras) and Prof. R. Balasubramanian (Institute of Mathematical Sciences); the team at the Indian Institute of Science who maintained the conference website; and the local organizing team at the Indian Institute of Technology, Madras, for their sincere hard work and for the local organization matters for the conference. We are especially grateful to our sponsors for their generous support of the conference. We would also like to express our appreciation to Springer for their active cooperation and timely production of the proceedings.

Finally, we would like to thank all the authors who submitted their work to INDOCRYPT 2017, and all the attendees. Without your spirited participation, the conference would not be a success. We hope you enjoy the proceedings of this year's INDOCRYPT conference.

December 2017

Nigel P. Smart
Arpita Patra

INDOCRYPT 2017

18th International Conference on Cryptology in India

Chennai, India
10–13 December, 2017

General Chairs

C. Pandu Rangan Indian Institute of Technology Madras, India
R. Balasubramanian Institute of Mathematical Sciences, India

Program Chairs

Arpita Patra Indian Institute of Science, India
Nigel P. Smart University of Bristol, UK

Program Committee

Adeline CNRS/IRISA, France
 Roux-Langlois
Aggelos Kiayias University of Edinburgh, UK
Alessandra Scafuro North Carolina State University, USA
Andrey Bogdanov Technical University of Denmark, Denmark
Anja Lehmann IBM Research - Zurich, Switzerland
Bart Preneel KU Leuven, Belgium
Benny Pinkas Bar-Ilan University, Israel
Bhavana Kanukurthi Indian Institute of Science, India
Carmit Hazay Bar-Ilan University, Israel
Chris Brzuska Hamburg University, Germany
Christophe Petit Oxford University, UK
Debdeep Indian Institute of Technology Kharagpur, India
 Mukhopadhyay
Dennis Hofheinz Karlsruhe Institute of Technology, Germany
Francois-Xavier Catholic University of Louvain, Belgium
 Standaert
Georg Fuchsbauer ENS Paris, France
Giuseppe Persiano University of Salerno, Italy
Goutam Paul Indian Statistical Institute Kolkata, India
Helena Handschuh Rambus Cryptography Research and KU Leuven, Belgium
Itai Dinur Ben-Gurion University, Israel
Jesper Buus Nielsen Aarhus University, Denmark
Jonathan Katz University of Maryland Park, USA
Joppe W. Bos NXP Semiconductors, Belgium

Kaoru Kurosawa	Ibaraki University, Japan
Kenny Paterson	Royal Holloway, University of London, UK
Krzysztof Pietrzak	IST Austria, Austria
Manoj Prabhakaran	Indian Institute of Technology Bombay, India
Marc Fischlin	Darmstadt University of Technology, Germany
Martin Albrecht	Royal Holloway, University of London, UK
Mike Rosulek	Oregon State, USA
Nishanth Chandran	Microsoft Research Bangalore, India
Ranjit Kumerasan	Microsoft Research Redmond, USA
Rosario Gennaro	City University New York, USA
Shweta Agrawal	Indian Institute of Technology Madras, India
Somitra Kr. Sanadhya	Ashoka University, India
Takahiro Matsuda	AIST, Japan
Tancrède Lepoint	SRI, USA
Thomas Johansson	Lund University, Sweden
Thomas Schneider	Darmstadt University of Technology, Germany
Vipul Goyal	Carnegie Mellon University, USA
Yu Sasaki	NTT Secure Platform Laboratories, Japan

External Reviewers

Alessandro Amadori	Shay Gueron	Sai Lakshmi Bhavana
Nuttapong Attrapadung	Mike Hamburg	Obbattu
Shi Bai	Mike Hutter	Sikhar Patranabis
Iddo Bentov	Ryo Kikuchi	Alice Pellet-Mary
Pauline Bert	Ágnes Kiss	Ben Pring
Begül Bilgin	Fuyuki Kitagawa	Chen Qian
Estuardo Alpirez Bock	Abhishek Kumar	Peter Rindal
Suvradip Chakraborty	Prabhat Kushwaha	Debapriya Basu Roy
Liqun Chen	Chaoyun Li	Yusuke Sakai
Madhuparna Das	Pierre Loidreau	Sruthi Sekar
Nayana Das	Monosij Maitra	Aishwarya
Nilanjan Datta	Daniel Malinowski	Thiruvengadam
Daniel Demmler	Alex Malozemoff	Yan Bo Ti
Orr Dunkelman	Bimal Mandal	Prashant Vasudevan
Dario Fiore	Sogol Mazaheri	Christian Weinert
Ran Gelles	James McKee	Kazuki Yoneyama
Mohona Ghosh	Michele Minelli	
Dahmun Goudarzi	Michael Naehrig	

Post-quantum Cryptography Using Module Lattices (Invited Talk)

Tancrède Lepoint 

SRI International, New York, USA

Recent advances in quantum computing and the announcement by the National Institute of Standards and Technology (NIST) to define new standards for digital-signature, encryption, and key-establishment protocols, spurred on the design and analysis of many post-quantum cryptographic schemes. One of the most efficient quantum-resilient alternatives for the above basic primitives is that of lattice cryptography.

Many lattice cryptography schemes are based on the *learning-with-error* problem over a ring R_q . Fix size parameters $k, \ell \geq 1$ and an ‘error’ probability χ on R_q . Let $A_{s,\chi}$ on $R_q^\ell \times R_q$ be the probability distribution obtained by choosing a vector $a \in R_q^\ell$ uniformly at random, choosing $e \in R_q$ according to χ , and outputting $(a, \langle a, s \rangle + e)$ where additions are performed in R_q . In the (decision) learning-with-error problem, the goal is to distinguish $A_{s,\chi}$, for a uniformly random secret $s \in R_q^\ell$, from the uniform distribution over $R_q^\ell \times R_q$. Most past works have described digital signature schemes, encryption schemes, and key encapsulation mechanisms in one of two ways. Either they set the parameters $k = \ell = 1$ and $R_q = \mathbf{Z}_q[x]/(x^t + 1)$ or they set $k, \ell > 1$ and $R_q = \mathbf{Z}_q$. The former choice results in schemes based on the hardness of the Ring-LWE and Ring-SIS problems (or the NTRU problem), while the latter choice of parameters results in schemes based on the LWE and SIS problems. In this talk, we consider the general case where $k, \ell \geq 1$ and $R = \mathbf{Z}_q[x]/(x^t + 1)$: this case results in schemes based on the Module-LWE and Module-SIS problems [3].

First, we explain how “module lattices” enable to design cryptographic primitives that are not only simple to implement securely, conservatively designed, and have a small memory footprint, but are *modular*, i.e., easily enable to vary security while keeping the same core operations. Then, we present *Kyber* [1], a key encapsulation mechanism, and *Dilithium* [2], a digital signature, part of CRYSTALS—*Cryptographic Suite for Algebraic Lattices*—, a portfolio of cryptographic primitives based on the Module-LWE and Module-SIS hardness assumptions submitted to the NIST call for post-quantum standards.

References

1. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: CRYSTALS - kyber: a cca-secure module-lattice-based KEM. *IACR Cryptology (2017)*. ePrint Archive, 2017:634
2. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, C.: CRYSTALS - dilithium: digital signatures from module lattices. *IACR Cryptology (2017)*. ePrint Archive, 2017:633
3. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. *Des. Codes Crypt.* **75**(3), 565–599 (2015)

Contents

Recent Advances in Function and Homomorphic Secret Sharing (Invited Talk)	1
<i>Elette Boyle</i>	
A Note on Ring-LWE Security in the Case of Fully Homomorphic Encryption	27
<i>Guillaume Bonnoron and Caroline Fontaine</i>	
Architecture Level Optimizations for Kummer Based HECC on FPGAs.	44
<i>Gabriel Gallin, Turku Ozlum Celik, and Arnaud Tisserand</i>	
Bricklayer Attack: A Side-Channel Analysis on the ChaCha Quarter Round	65
<i>Alexandre Adomnicai, Jacques J. A. Fournier, and Laurent Masson</i>	
CCA-secure Predicate Encryption from Pair Encoding in Prime Order Groups: Generic and Efficient.	85
<i>Sanjit Chatterjee, Sayantan Mukherjee, and Tapas Pandit</i>	
Cold Boot Attacks on NTRU	107
<i>Kenneth G. Paterson and Ricardo Villanueva-Polanco</i>	
Differential Cryptanalysis of 18-Round PRIDE.	126
<i>Virginie Lallemand and Shahram Rasoolzadeh</i>	
DSA Signing Key Recovery with Noisy Side Channels and Variable Error Rates	147
<i>Jiji Angel, R. Rahul, C. Ashokkumar, and Bernard Menezes</i>	
Efficient Construction of Diamond Structures	166
<i>Ariel Weizmann, Orr Dunkelman, and Simi Haber</i>	
Efficient Optimal Ate Pairing at 128-Bit Security Level.	186
<i>Md. Al-Amin Khandaker, Yuki Nanjo, Loubna Ghammam, Sylvain Duquesne, Yasuyuki Nogami, and Yuta Kodera</i>	
Fast Scalar Multiplication for Elliptic Curves over Binary Fields by Efficiently Computable Formulas	206
<i>Saud Al Musa and Guangwu Xu</i>	
Field Lifting for Smaller UOV Public Keys	227
<i>Ward Beullens and Bart Preneel</i>	

Gabidulin Matrix Codes and Their Application to Small Ciphertext
Size Cryptosystems 247
Thierry P. Berger, Philippe Gaborit, and Olivier Ruatta

Lightweight Design Choices for LED-like Block Ciphers 267
*Sumanta Sarkar, Habeeb Syed, Rajat Sadhukhan,
and Debdeep Mukhopadhyay*

Looting the LUTs: FPGA Optimization of AES and AES-like Ciphers
for Authenticated Encryption 282
Mustafa Khairallah, Anupam Chattopadhyay, and Thomas Peyrin

Improved Differential Cryptanalysis on Generalized Feistel Schemes 302
Ivan Tjuawinata, Tao Huang, and Hongjun Wu

Improvements for Gate-Hiding Garbled Circuits 325
Mike Rosulek

Recovering Short Generators of Principal Fractional Ideals
in Cyclotomic Fields of Conductor $p^z q^\beta$ 346
Patrick Holzer, Thomas Wunderer, and Johannes A. Buchmann

Revisiting a Masked Lookup-Table Compression Scheme 369
Srinivas Vivek

Several Masked Implementations of the Boyar-Peralta AES S-Box 384
Ashrutit Ghoshal and Thomas De Cnudde

Author Index 403

Recent Advances in Function and Homomorphic Secret Sharing

(Invited Talk)

Elette Boyle^(✉)

IDC Herzliya, Herzliya, Israel
eboyle@alum.mit.edu

Abstract. Function Secret Sharing (FSS) and Homomorphic Secret Sharing (HSS) are two extensions of standard secret sharing, which support rich forms of homomorphism on secret shared values.

- An m -party FSS scheme for a given function family \mathcal{F} enables splitting a function $f : \{0, 1\}^n \rightarrow \mathbb{G}$ from \mathcal{F} (for Abelian group \mathbb{G}) into m succinctly described functions f_1, \dots, f_m such that strict subsets of the f_i hide f , and $f(x) = f_1(x) + \dots + f_m(x)$ for every input x .
- An m -party HSS is a dual notion, where an input x is split into shares x^1, \dots, x^m , such that strict subsets of x^i hide x , and one can recover the evaluation $P(x)$ of a program P on x given homomorphically evaluated share values $\text{Eval}(x^1, P), \dots, \text{Eval}(x^m, P)$.

In the last few years, many new constructions and applications of FSS and HSS have been discovered, yielding implications ranging from efficient private database manipulation and secure computation protocols, to worst-case to average-case reductions.

In this treatise, we introduce the reader to the background required to understand these developments, and give a roadmap of recent advances (up to October 2017).

1 Introduction

A secret sharing scheme [38] enables a dealer holding a secret s to randomly split s into m shares, such that certain subsets of the shares can be used to reconstruct the secret and others reveal nothing about it. The simplest type of secret sharing is *additive secret sharing*, where the secret is an element of an Abelian group \mathbb{G} , it can be reconstructed by adding all m shares, and every subset of $m - 1$ shares reveals nothing about the secret. A useful feature of this secret sharing scheme is that it is (linearly) *homomorphic*, in the sense that if m parties hold shares of many secrets, they can locally compute shares of the sum of all secrets. This feature of additive secret sharing (more generally, linear secret sharing) is useful for many cryptographic applications.

Supported in part by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069, and ERC Grant no. 307952.

A line of recent works [6–11, 28] has investigated secret sharing schemes which support *richer* classes of homomorphism. In this survey, we present recent developments in the following (closely related) natural extensions of additive secret sharing:

- **Function Secret Sharing (FSS)** [6]. Suppose we are given a class \mathcal{F} of efficiently computable and succinctly described functions $f : \{0, 1\}^n \rightarrow \mathbb{G}$. Is it possible to split an arbitrary function $f \in \mathcal{F}$ into m functions f_1, \dots, f_m such that: (1) each f_i is described by a short key k_i that enables its efficient evaluation, (2) strict subsets of the keys completely hide f , and (3) $f(x) = \sum_{i=1}^m f_i(x)$ (on every input x)? We refer to a solution to this problem as a *function secret sharing* (FSS) scheme for \mathcal{F} .
- **Homomorphic Secret Sharing (HSS)** [8]. A (m -party) HSS scheme for class of programs¹ \mathcal{P} randomly splits an input x into shares² (x^1, \dots, x^m) such that: (1) each x^i is polynomially larger than x , (2) subsets of shares x^i hide x , and (3) there exists a polynomial-time local evaluation algorithm Eval such that for any “program” $P \in \mathcal{P}$ (e.g., a boolean circuit, formula or branching program), the output $P(x)$ can be efficiently reconstructed from $\text{Eval}(x^1, P), \dots, \text{Eval}(x^m, P)$.

FSS can be thought of as a dual notion of HSS, where the roles of the function and input are reversed: FSS considers the goal of secret sharing a function f (represented by a program) in a way that enables compact evaluation on any given input x via local computation on the shares of f , and HSS considers the goal of secret sharing an input x in a way that enables compact evaluation of any given function f via local computation on the shares of x .

While any FSS scheme can be viewed as an HSS scheme for a suitable class of programs and vice versa, the notions of “FSS for \mathcal{P} ” and “HSS for \mathcal{P} ” for a given program class \mathcal{P} are *not* identical, in that FSS allows the share size to grow with the size of the programs $P \in \mathcal{P}$, whereas HSS restricts share size to grow with the size of the *input* to P .

In addition, HSS admits a natural multi-input variant (where secrets originating from different parties can be homomorphically evaluated on together), whereas in FSS the secret function always originates from a single source.

In different applications and examples, FSS or HSS perspective is more natural.

Computational security. Unlike secret sharing with basic linear homomorphism, it can be shown that most nontrivial FSS and HSS cannot provide information theoretic hiding [6, 11, 28]. For example, even for simple classes \mathcal{F} (such

¹ Function vs. program: Note that in FSS we will consider simple classes of functions where each function has a unique description, whereas in HSS we consider functions with many programs computing it. For this reason we refer to “function” for FSS and “program” for HSS.

² f_i vs. x^i : We maintain the subscript/superscript conventions of existing works (primarily [6, 11]). Note that superscript notation is used in HSS where one can consider shares of multiple inputs, $x_j \mapsto (x_j^1, \dots, x_j^m)$.

as the class of point functions), the best possible solution is to additively share the truth-table representation of f , whose shares consist of 2^n group elements. But if one considers a *computational* notion of hiding, then there are no apparent limitations to what can be done for polynomial-time computable f . This is what we refer to when we speak of FSS/HSS.

Homomorphic secret sharing vs. fully homomorphic encryption. HSS can be viewed as a relaxed version of fully homomorphic encryption (FHE) [26, 37], where instead of a single party homomorphically evaluating on encrypted data, we allow homomorphic evaluation to be distributed among two parties who do not interact with each other. As in the case of FHE, we require that the output of `Eval` be *compact* in the sense that its length depends only on the output length $|P(x)|$ but not on the size of P . But in fact, a unique feature of HSS that distinguishes it from traditional FHE is that the output representation can be *additive*. E.g., we can achieve $\text{Eval}(x^0, P) + \text{Eval}(x^1, P) = P(x) \bmod \beta$ for some positive integer $\beta \geq 2$ that can be chosen arbitrarily. This enables an ultimate level of compactness and efficiency of reconstruction that is impossible to achieve via standard FHE. For instance, if P outputs a single bit and $\beta = 2$, then the output $P(x)$ is reconstructed by taking the exclusive-or of two bits.

Other related notions. We note that other forms of secret sharing of functions and homomorphic secret sharing have been considered in the literature. An initial study of secret sharing homomorphisms is due to Benaloh [4], who presented constructions and applications of additively homomorphic secret sharing schemes. Further exploration of computing on secret shared data took place in [1]. Secret sharing of functions has appeared in the context of threshold cryptography (cf. [19, 20]). However, these other notions either apply only to very specific function classes that enjoy homomorphism properties compatible with the secret sharing, or alternatively they do not require a simple (e.g., additive) representation of the output which is essential for the applications we consider.

1.1 This Survey

The aim of this document is to serve as a centralized resource for FSS and HSS, providing sufficient background to approach existing papers, and appropriate references of where to look for further details. In what follows, we present:

- Formal definitions. This includes a discussion on different choices of reconstruction procedures (and why we focus on linear reconstruction), an application-targeted definition of FSS, and a broader theory-oriented definition of HSS.
- Constructions. A guide to existing constructions within the literature, and an overview of two specific constructions: FSS for point functions from one-way functions [9], and HSS for branching programs (with $1/\text{poly}$ error) from the Decisional Diffie-Hellman assumption [8].
- Applications. Discussion on implications and applications of FSS and HSS and appropriate pointers.

Low-end vs. high-end. A recurring theme throughout the survey is that constructions and applications fall predominantly into two categories:

- “Low-end” lightweight constructions for simple function classes.
- “High-end” powerful constructions for broad function classes.

The former refers to constructions from symmetric-key primitives (in particular, one-way functions), sits closer to current practical applications, and is most frequently associated with the FSS formulation. The latter includes constructions from public-key primitives, yields powerful feasibility implications, and is most frequently associated with the HSS formulation. We will present results from this perspective.

2 Definitions

At their core, FSS/HSS are secret sharing schemes, and as such demand two central properties: (1) Correctness, dictating the appropriate homomorphic evaluation guarantees, and (2) Privacy, requiring that subsets of shares do not reveal the original secret.

When defining FSS/HSS, there are a handful of different choices to be made that result in slightly shifted notions of varying generality. We choose two such definitions to present:

1. FSS targeted definition, most directly in line with practical applications.
2. HSS general definition, which can be instantiated to capture different notions from the literature, including those of theoretical works such as [8, 10, 11] as well as the FSS definition from above.

Before jumping to these definitions, we begin with some basic notation and a discussion on different choices of output decoding structure.

2.1 Basic Notation

We denote the security parameter by λ .

Modeling function families. A *function family* is defined by a pair $\mathcal{F} = (P_{\mathcal{F}}, E_{\mathcal{F}})$, where $P_{\mathcal{F}} \subseteq \{0, 1\}^*$ is an infinite collection of function descriptions \hat{f} , and $E_{\mathcal{F}} : P_{\mathcal{F}} \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ is a polynomial-time algorithm defining the function described by \hat{f} . Concretely, each $\hat{f} \in P_{\mathcal{F}}$ describes a corresponding function $f : D_f \rightarrow R_f$ defined by $f(x) = E_{\mathcal{F}}(\hat{f}, x)$. We assume by default that $D_f = \{0, 1\}^n$ for a positive integer n (though will sometimes consider inputs over non-binary alphabets) and always require R_f to be a finite Abelian group, denoted by \mathbb{G} . When there is no risk of confusion, we will sometimes write f instead of \hat{f} and $f \in \mathcal{F}$ instead of $\hat{f} \in P_{\mathcal{F}}$. We assume that \hat{f} includes an explicit description of both D_f and R_f as well as a size parameter $S_{\hat{f}}$.

2.2 Discussion on Output Decoding Structure

One can consider FSS/HSS with respect to many choices of output decoding structure: that is, the procedure used to combine homomorphically evaluated shares into the desired output. Based on the structure of the chosen decoding process, the corresponding scheme will have very different properties: more complex decoding procedures open the possibility of achieving FSS/HSS for more general classes of functions, but place limits on the applicability of the resulting scheme. Many choices for the structure of the output decoding function yield uninteresting notions, as we now discuss (following [6]). For convenience, we adopt the language of FSS.

Arbitrary reconstruction. Consider, for example, FSS with *no restriction* on the reconstruction procedure for parties’ output shares. Such wide freedom renders the notion non-meaningfully trivial. Indeed, for any efficient function family \mathcal{F} , one could generate FSS keys for a secret function $f \in \mathcal{F}$ simply by sharing a description of f *interpreted as a string*, using a standard secret sharing scheme. The evaluation procedure on any input x will simply output x together with the party’s share of f , and the decoding procedure will first reconstruct the description of f , and then compute and output the value $f(x)$.

This construction satisfies correctness and security as described informally above (indeed, each party’s key individually reveals no information on f). But, the scheme clearly leaves much to be desired in terms of utility: From just one evaluation, the entire function f is revealed to whichever party receives and reconstructs these output shares. At such point, the whole notion of function secret sharing becomes moot.

“Function-private” output shares. Instead, from a function secret sharing scheme, one would hope that parties’ output shares $f_i(x)$ for input x do not reveal more about the secret function f than is necessary to determine $f(x)$. That is, we may impose a “function privacy” requirement on the reconstruction scheme, requiring that pairs of parties’ output shares for each input x can be simulated given just the corresponding outputs $f(x)$.

This requirement is both natural and beneficial, but by itself still allows for undesired constructions. For example, given a secret function f , take one FSS key to be a *garbled circuit* of f , and the second key as the information that enables translating inputs x to garbled input labels. This provides a straightforward function-private solution for one output evaluation, and can easily be extended to the many-output case by adding shared secret randomness to the parties’ keys.³ Yet this construction (and thus definition) is unsatisfying: although the evaluate output shares $f_i(x)$ now hide f , their size is massive—for every output, comparable to a copy of f itself. (Further, this notion does not give any cryptographic power beyond garbled circuits.)

³ Namely, for each new x , the parties will first use their shared randomness to coordinately rerandomize the garbled circuit of f and input labels, respectively.

Succinct, function-private output shares. We thus further restrict the scheme, demanding additionally that output shares be *succinct*: i.e., comparable in size to the function output.

This definition already captures a strong, interesting primitive. For example, as described in Sect. 4, achieving such an FSS scheme for general functions implies a form of communication-efficient secure multi-party computation. Additional lower bounds on this notion are shown in [11]. However, there is one final property that enables an important class of applications, but which is not yet guaranteed: a notion of *share compressibility*.

More specifically: One of the central application regimes of FSS [6, 9, 28] is enabling communication-efficient secure (m -server) Private Information Retrieval (PIR). Intuitively, to privately recover an item x_i from a database held by both servers, one can generate and distribute a pair of FSS keys encoding a point function f_i whose only nonzero output is at secret location i . Each server then responds with a *single* element, computed as the weighted sum of each data item x_j with the server’s output share of the evaluation $f_i(x_j)$. Correctness of the DPF scheme implies that the xor of the two servers’ replies is precisely the desired data item x_i , while security guarantees the servers learn nothing about the index i . But most importantly, the linear structure of the DPF reconstruction enabled the output shares pertaining to all the different elements of the database to be *compressed* into a single short response.

On the other hand, consider, for example, the PIR scenario but where the servers instead hold shares of the function f_i with respect to a *bitwise AND* reconstruction of output shares in the place of xor/addition. Recovery of the requested data item x_i now implies computing set intersection—and thus requires communication complexity equal to the size of the database [34]! We thus maintain the crucial property that output shares can be combined and compressed in a meaningful way. To do so, we remain in stride with the *linearity* of output share decoding.

Primary focus: linear share decoding. We focus predominantly on the setting of FSS where the output decoder is a *linear function* of parties’ shares. That is, we assume the output shares $f_i(x)$ lie within an Abelian group \mathbb{G} and consider a decoding function $\text{Dec} : \mathbb{G}^m \rightarrow \mathbb{G}$ linear in \mathbb{G} . This clean, intuitive structure in fact provides the desired properties discussed above: Linearity of reconstruction provides convenient share *compressibility*. Output shares must themselves be elements of the function output space, immediately guaranteeing share *succinctness*. And as shown in [6], the linear reconstruction in conjunction with basic key security directly implies *function privacy*. Unless otherwise specified we will implicitly take an “FSS scheme” (or HSS) to be one with a linear reconstruction procedure.

2.3 Function Secret Sharing: Targeting Applications

We next present a targeted definition of FSS, which lies most in line with the use of FSS within current practical applications. The definition follows [9], extending

the original definition from [6] by allowing a general specification of allowable *leakage*: i.e., partial information about the function that can be revealed.

Recall in the language of FSS, we consider a client holding a secret function $f \in \mathcal{F}$ who splits f into shares f_i supporting homomorphic evaluation on inputs x in the domain of f . We use notation of the shares f_i described by keys k_i .

Modeling leakage. We capture the allowable leakage by a function $\text{Leak} : \{0, 1\}^* \rightarrow \{0, 1\}^*$, where $\text{Leak}(f)$ is interpreted as the partial information about f that can be leaked. When Leak is omitted it is understood to output the input domain D_f and the output domain R_f . This will be sufficient for most classes considered; for some classes, one also needs to leak the size S_f . But, one can consider more general choices of Leak , which allow a tradeoff between efficiency/feasibility and revealed information. (E.g., the construction of FSS for decision trees in [9] leaks the topology of the tree but hides the labels; see Sect. 3.)

Definition 1 (FSS: Syntax). *An m -party function secret sharing (FSS) scheme is a pair of algorithms $(\text{Gen}, \text{Eval})$ with the following syntax:*

- $\text{Gen}(1^\lambda, \hat{f})$ is a PPT key generation algorithm, which on input 1^λ (security parameter) and $\hat{f} \in \{0, 1\}^*$ (description of a function f) outputs an m -tuple of keys (k_1, \dots, k_m) . We assume that \hat{f} explicitly contains an input length 1^n , group description \mathbb{G} , and size parameter.
- $\text{Eval}(i, k_i, x)$ is a polynomial-time evaluation algorithm, which on input $i \in [m]$ (party index), k_i (key defining $f_i : \{0, 1\}^n \rightarrow \mathbb{G}$) and $x \in \{0, 1\}^n$ (input for f_i) outputs a group element $y_i \in \mathbb{G}$ (the value of $f_i(x)$, the i -th share of $f(x)$).

When m is omitted, it is understood to be 2.

Definition 2 (FSS: Requirements). *Let $\mathcal{F} = (P_{\mathcal{F}}, E_{\mathcal{F}})$ be a function family and $\text{Leak} : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a function specifying the allowable leakage. Let m (number of parties) and t (secrecy threshold) be positive integers. An m -party t -secure FSS for \mathcal{F} with leakage Leak is a pair $(\text{Gen}, \text{Eval})$ as in Definition 1, satisfying the following requirements.*

- **Correctness:** For all $\hat{f} \in P_{\mathcal{F}}$ describing $f : \{0, 1\}^n \rightarrow \mathbb{G}$, and every $x \in \{0, 1\}^n$, if $(k_1, \dots, k_m) \leftarrow \text{Gen}(1^\lambda, \hat{f})$ then $\Pr[\sum_{i=1}^m \text{Eval}(i, k_i, x) = f(x)] = 1$.
- **Secrecy:** For every set of corrupted parties $S \subset [m]$ of size t , there exists a PPT algorithm Sim (simulator), such that for every sequence $\hat{f}_1, \hat{f}_2, \dots$ of polynomial-size function descriptions from $P_{\mathcal{F}}$, the outputs of the following experiments Real and Ideal are computationally indistinguishable:

- $\text{Real}(1^\lambda) : (k_1, \dots, k_m) \leftarrow \text{Gen}(1^\lambda, \hat{f}_\lambda)$; Output $(k_i)_{i \in S}$.
- $\text{Ideal}(1^\lambda) : \text{Output } \text{Sim}(1^\lambda, \text{Leak}(\hat{f}_\lambda))$.

When Leak is omitted, it is understood to be the function $\text{Leak}(\hat{f}) = (1^n, S_{\hat{f}}, \mathbb{G})$ where 1^n , $S_{\hat{f}}$, and \mathbb{G} are the input length, size, and group description contained in \hat{f} . When t is omitted it is understood to be $m - 1$.

A useful instance of FSS, introduced by Gilboa and Ishai [28], is a *distributed point function* (DPF). A DPF can be viewed as a 2-party FSS for the function class \mathcal{F} consisting of all point functions, namely all functions $f : \{0, 1\}^n \rightarrow \mathbb{G}$ that evaluate to 0 on all but at most one input.

Definition 3 (Distributed Point Function). A point function $f_{\alpha, \beta}$, for $\alpha \in \{0, 1\}^n$ and $\beta \in \mathbb{G}$, is defined to be the function $f : \{0, 1\}^n \rightarrow \mathbb{G}$ such that $f(\alpha) = \beta$ and $f(x) = 0$ for $x \neq \alpha$. We will sometimes refer to a point function with $|\beta| = 1$ (resp., $|\beta| > 1$) as a single-bit (resp., multi-bit) point function. A Distributed Point Function (DPF) is an FSS for the family of all point functions, with the leakage $\text{Leak}(\hat{f}) = (1^n, \mathbb{G})$.

A concrete security variant. For the purpose of describing and analyzing some FSS constructions, it is sometimes convenient (e.g., in [9]) to consider a *finite* family \mathcal{F} of functions $f : D_f \rightarrow R_f$ sharing the same (fixed) input domain and output domain, as well as a fixed value of the security parameter λ . We say that such a finite FSS scheme is (T, ϵ) -secure if the computational indistinguishability requirement in Definition 2 is replaced by (T, ϵ) -indistinguishability, namely any size- T circuit has at most an ϵ advantage in distinguishing between **Real** and **Ideal**. When considering an infinite collection of such finite \mathcal{F} , parameterized by the input length n and security parameter λ , we require that **Eval** and **Sim** be each implemented by a (uniform) PPT algorithm, which is given 1^n and 1^λ as inputs.

2.4 Homomorphic Secret Sharing: A General Definition

Recall that HSS is a dual form of FSS. We now consider more general multi-input HSS schemes that support a compact evaluation of a function F on shares of inputs x_1, \dots, x_n that originate from different clients. More concretely, each client i randomly splits its input x_i between m servers using the algorithm **Share**, so that x_i is hidden from any t colluding servers (we assume $t = m - 1$ by default). Each server j applies a local evaluation algorithm **Eval** to its share of the n inputs, and obtains an output share y^j . The output $F(x_1, \dots, x_n)$ is reconstructed by applying a decoding algorithm **Dec** to the output shares (y^1, \dots, y^m) . To avoid triviality, we consider various restrictions on **Dec** that force it to be “simpler” than direct computation of F .

Finally, for some applications it is useful to let F and **Eval** take an additional input x_0 that is known to all servers. This is necessary for a meaningful notion of single-input HSS (with $n = 1$) [8], and function secret sharing [6, 9]. Typically, the extra input x_0 will be a description of a function f applied to the input of a single client, e.g., a description of a circuit, branching program, or low-degree polynomial. For the case of FSS, the (single) client’s input is a description of a program and the additional input x_0 corresponds to a domain element.

We now give our formal definition of general HSS. We give a definition in the plain model; this definition can be extended in a natural fashion to settings

with various forms of setup (e.g., common public randomness or a public-key infrastructure, as considered in [10]). We follow the exposition of [11]. Recall subscripts denote input (client) id and superscripts denote share (server) id.

Definition 4 (HSS). *An n -client, m -server, t -secure homomorphic secret sharing scheme for a function $F : (\{0,1\}^*)^{n+1} \rightarrow \{0,1\}^*$, or (n,m,t) -HSS for short, is a triple of PPT algorithms (Share, Eval, Dec) with the following syntax:*

- **Share**($1^\lambda, i, x$): *On input 1^λ (security parameter), $i \in [n]$ (client index), and $x \in \{0,1\}^*$ (client input), the sharing algorithm Share outputs m input shares, (x^1, \dots, x^m) .*
- **Eval**($j, x_0, (x_1^j, \dots, x_n^j)$): *On input $j \in [m]$ (server index), $x_0 \in \{0,1\}^*$ (common server input), and x_1^j, \dots, x_n^j (j th share of each client input), the evaluation algorithm Eval outputs $y^j \in \{0,1\}^*$, corresponding to server j 's share of $F(x_0; x_1, \dots, x_n)$.*
- **Dec**(y^1, \dots, y^m): *On input (y^1, \dots, y^m) (list of output shares), the decoding algorithm Dec computes a final output $y \in \{0,1\}^*$.*

The algorithms (Share, Eval, Dec) should satisfy the following correctness and security requirements:

- **Correctness:** *For any $n+1$ inputs $x_0, \dots, x_n \in \{0,1\}^*$,*

$$\Pr \left[\forall i \in [n] (x_i^1, \dots, x_i^m) \leftarrow \text{Share}(1^\lambda, i, x_i) : \text{Dec}(y^1, \dots, y^m) = F(x_0; x_1, \dots, x_n) \right] = 1.$$

Alternatively, in a statistically correct HSS the above probability is at least $1 - \mu(\lambda)$ for some negligible μ and in a δ -correct HSS (or δ -HSS for short) it is at least $1 - \delta - \mu(\lambda)$, where the error parameter δ is given as an additional input to Eval and the running time of Eval is allowed to grow polynomially with $1/\delta$.

- **Security:** *Consider the following semantic security challenge experiment for corrupted set of servers $T \subset [m]$:*

1: *The adversary gives challenge index and inputs $(i, x, x') \leftarrow \mathcal{A}(1^\lambda)$, with $|x| = |x'|$.*

2: *The challenger samples $b \leftarrow \{0,1\}$ and $(x^1, \dots, x^m) \leftarrow \text{Share}(1^\lambda, i, \tilde{x})$, where $\tilde{x} = \begin{cases} x & \text{if } b = 0 \\ x' & \text{else} \end{cases}$.*

3: *The adversary outputs a guess $b' \leftarrow \mathcal{A}((x^j)_{j \in T})$, given the shares for corrupted T .*

Denote by $\text{Adv}(1^\lambda, \mathcal{A}, T) := \Pr[b = b'] - 1/2$ the advantage of \mathcal{A} in guessing b in the above experiment, where probability is taken over the randomness of the challenger and of \mathcal{A} .

For circuit size bound $S = S(\lambda)$ and advantage bound $\alpha = \alpha(\lambda)$, we say that an (n,m,t) -HSS scheme $\Pi = (\text{Share}, \text{Eval}, \text{Dec})$ is (S, α) -secure if for all $T \subset [m]$ of size $|T| \leq t$, and all non-uniform adversaries \mathcal{A} of size $S(\lambda)$, we have $\text{Adv}(1^\lambda, \mathcal{A}, T) \leq \alpha(\lambda)$. We say that Π is:

- computationally secure if it is $(S, 1/S)$ -secure for all polynomials S ;
- statistically α -secure if it is (S, α) -secure for all S ;
- statistically secure if it is statistically α -secure for some negligible $\alpha(\lambda)$;
- perfectly secure if it is statistically 0-secure.

Remark 1 (Unbounded HSS). Definition 4 treats the number of inputs n as being fixed. We can naturally consider an unbounded multi-input variant of HSS where F is defined over arbitrary sequences of inputs x_i , and the correctness requirement is extended accordingly. We denote this flavor of multi-input HSS by $(*, m, t)$ -HSS. More generally, one can allow all three parameters n, m, t to be flexible, treating them as inputs of the three algorithms Share, Eval, Dec.

Remark 2 (Comparing to FSS Definition). Function secret sharing (FSS) as per Definition 2 can be cast in the definition above as $(1, m)$ -HSS for the universal function $F(x; P) = P(x)$, where $P \in \mathcal{P}$ is a program given as input to the client and x is the common server input.

Note the security requirement for HSS in Definition 4 is expressed as an indistinguishability guarantee, whereas the FSS definition from the previous section (Definition 2) referred instead to efficient simulation given leakage on the secret data. However, the two flavors are equivalent for every function family \mathcal{F} and leakage function Leak for which Leak can be efficiently inverted; that is, given $\text{Leak}(\hat{f})$ one can efficiently find \hat{f}' such that $\text{Leak}(\hat{f}') = \text{Leak}(\hat{f})$. Such an inversion algorithm exists for all instances of \mathcal{F} and Leak considered in existing works.

As discussed, Definition 4 can be trivially realized by Eval that computes the identity function. To make HSS useful, we impose two types of requirements on the decoding algorithm.

Definition 5 (Additive and compact HSS). We say that an (n, m, t) -HSS scheme $\Pi = (\text{Share}, \text{Eval}, \text{Dec})$ is:

- Additive if Dec outputs the exclusive-or of the m output shares. Alternatively, if Dec interprets its m arguments as elements of an Abelian group \mathbb{G} (instead of bit strings), and outputs their sum in \mathbb{G} .⁴
- Compact if the length of the output shares is sublinear in the input length when the inputs are sufficiently longer than the security parameter. Concretely:
 - We say that Π is $g(\lambda, \ell)$ -compact if for every λ, ℓ , and inputs $x_0, x_1, \dots, x_n \in \{0, 1\}^\ell$, the length of each output share obtained by applying Share with security parameter λ and then Eval is at most $g(\lambda, \ell)$.
 - We say that Π is compact if it is $g(\lambda, \ell)$ -compact for g that satisfies the following requirement: There exists a polynomial $p(\cdot)$ and sublinear function $g'(\ell) = o(\ell)$ such that for any λ and $\ell \geq p(\lambda)$ we have $g(\lambda, \ell) \leq g'(\ell)$.

In the case of perfect security or statistical α -security with constant α , we eliminate the parameter λ and refer to Π as being $g(\ell)$ -compact.

⁴ In this case, we think of the function F and all HSS algorithms Share, Eval, Dec as implicitly receiving a description of \mathbb{G} as an additional input.

Remark 3 (Other notions of compactness). One could alternatively consider a stronger notion of compactness, requiring that the length of each output share is of the order of the output length (whereas Definition 5 requires merely for it to be sublinear in the input size). Every additive HSS scheme satisfies this notion. HSS schemes that satisfy this notion but are not additive were used in the context of private information retrieval and locally decodable codes in [2]. A different way of strengthening the compactness requirement is by restricting the *computational complexity* of Dec, e.g., by requiring it to be quasi-linear in the length of the output. See Sect. 4.1 (worst-case to average-case reductions) for motivating applications.

Remark 4 (Special HSS Cases)

- We will sometimes be interested in additive HSS for a *finite function* F , such as the AND of two bits; this can be cast into Definition 4 by just considering an extension \hat{F} of F that outputs 0 on all invalid inputs. (Note that our notion of compactness is not meaningful for a finite F .)
- As noted above, the common server input x_0 is often interpreted as a “program” P from a class of programs \mathcal{P} (e.g., circuits or branching programs), and F is the universal function defined by $F(P; x_1, \dots, x_n) = P(x_1, \dots, x_n)$. We refer to this type of HSS as *HSS for the class* \mathcal{P} .

3 Constructions of FSS and HSS

FSS/HSS constructions as of the writing of this survey (October 2017) are as follows. Given complexity measures are with respect to n -bit inputs.

“Low End”: FSS from One-Way Functions

Here λ corresponds to a pseudorandom generator seed length, taken to be 128 bits in an AES-based implementation. Unless otherwise specified, for $m = 2$ servers.

- **Point functions** (“Distributed Point Functions”).

The class of point functions consists of those functions $f_{\alpha, \beta}$ which evaluate to β on input α and to 0 otherwise.

- Implicitly constructed in [15] with key size $O(2^{\epsilon n})$ bits for constant $\epsilon > 0$. Formally defined and constructed recursively with key size $O(n^{\log_2(3)} \lambda)$ bits, in [28]. Improved to $O(n\lambda)$ bits via tree-based solution in [6].
- Current best: Key size $\lambda + n(\lambda + 2) - \lfloor \log \lambda / |\beta| \rfloor$ bits, in [9].⁵

For $m > 2$ servers: nontrivial (but poor) key size $O(2^m 2^{n/2} \lambda)$ bits, in [6].

⁵ In particular: $\lambda + n(\lambda + 2)$ for λ -bit outputs, and $\lambda + n(\lambda + 2) - \lfloor \log \lambda \rfloor$ for 1-bit outputs.

- **Comparison and Intervals** [6,9].

The class of comparison functions consists of those functions f_a which output 1 on inputs x with $a < x$. Interval functions $f_{(a,b)}$ output 1 precisely for inputs x that lie within the interval $a < x < b$, and 0 otherwise.

Constructions follow a similar structure as DPFs. Best key size for comparison functions $n(\lambda + 3)$ bits, for interval functions $2n(\lambda + 3)$ bits [9].

- NC^0 **predicates** (i.e., functions with constant locality) [6].

For locality d , the key size grows as $O(\lambda \cdot n^d)$. For example, this includes bit-matching predicates that check a constant number of bits d .

- **Decision trees** [9].

A decision tree is defined by: (1) a tree topology, (2) variable labels on each node v (where the set of possible values of each variable is known), (3) value labels on each edge (the possible values of the originating variable), and (4) output labels on each leaf node.

In the construction of [9], the key size is roughly $\lambda \cdot |V|$ bits, where V is the set of nodes, and evaluation on a given input requires $|V|$ executions of a pseudorandom generator, and a comparable number of additions. The FSS is guaranteed to hide the secret edge value labels and leaf output labels, but (in order to achieve this efficiency) reveals the base tree topology and the identity of which variable is associated to each node.

Constant-dimensional intervals. A sample application of FSS for decision trees is constant d -dimensional interval queries: that is, functions $f(x_1, \dots, x_d)$ which evaluate to a selected nonzero value precisely when $a_i \leq x_i \leq b_i$ for some secret interval ranges $(a_i, b_i)_{i \in [d]}$. For n -bit inputs x_i , FSS for d -dimensional intervals can be obtained with key size and computation time $O(\lambda \cdot n^d)$. For small values of d , such as $d = 2$ for supporting a conjunction of intervals, this yields solutions with reasonably good concrete efficiency.

We observe that FSS constructions for the function classes above can be combined with server-side database operations, to emulate private database operations of richer function classes, such as Max/Min and top- k [40]. (See Sect. 4.2.)

“High End”: HSS from Public-Key Cryptography

- **Branching programs** (capturing logspace, NC^1), for 2-servers, with inverse-polynomial δ -correctness, from Decisional Diffie-Hellman (DDH) [8]. Evaluation runtime grows as $1/\delta$.

Heavily optimized versions of this construction are given in [7,10].

- **General circuits**, from Learning With Errors (LWE) [6,23].

More specifically, in the language of Definition 4: Additive (n, m) -HSS for arbitrary n, m and polynomial-size circuits can be obtained from the Learning With Errors (LWE) assumption, by a simple variation of the FSS construction from spooky encryption of [23] (more specifically, their techniques for obtaining 2-round MPC). See [11] for details.

(It was also previously shown how to achieve FSS for general circuits from subexponentially secure indistinguishability obfuscation in [6].)

Intuition of Constructions. In the following two subsections, we present high-level intuition behind two specific constructions: (1) the optimized OWF-based DPF of [9], and (2) the DDH-based δ -HSS for branching programs of [8].

3.1 Overview: Distributed Point Function from OWF

We give an intuitive description of the (2-party) distributed point function (DPF) ($\text{Gen}^\bullet, \text{Eval}^\bullet$) construction from [9] (following the text therein). Recall a DPF is an FSS scheme for the class of point functions $f_{\alpha,\beta} : \{0,1\}^n \rightarrow \mathbb{G}$ whose only nonzero evaluation is $f_{\alpha,\beta}(\alpha) = \beta$. For simplicity, consider the case of a DPF with a single-bit output $\mathbb{G} = \{0,1\}$ and $\beta = 1$.

Basic key structure. At a high level, each of the two DPF keys k_0, k_1 defines a GGM-style binary tree [29] with 2^n leaves, where the leaves are labeled by inputs $x \in \{0,1\}^n$. We will refer to a path from the root to a leaf labeled by x as the *evaluation path* of x , and to the evaluation path of the special input α as the *special evaluation path*. Each node v in a tree will be labeled by a string of length $\lambda + 1$, consisting of a *control bit* t and a λ -bit *seed* s , where the label of each node is fully determined by the label of its parent. The function Eval^\bullet will compute the labels of all nodes on the evaluation path to the input x , using the root label as the key, and output the control bit of the leaf.

Generating the keys. We would like to maintain the invariant that for each node outside the special path, the two labels (on the two trees) are identical, and for each node on the special path the two control bits are different and the two seeds are indistinguishable from being random and independent. Note that since the label of a node is determined by that of its parent, if this invariant is met for a node outside the special path then it is automatically maintained by its children. Also, we can easily meet the invariant for the root (which is always on the special path) by just explicitly including the labels in the keys. The challenge is to ensure that the invariant is maintained also when leaving the special path.

Towards describing the construction, it is convenient to view the two labels of a node as a mod-2 additive secret sharing of its label, consisting of shares $[t] = (t_0, t_1)$ of the control bit t and shares $[s] = (s_0, s_1)$ of the λ -bit seed s . That is, $t = t_0 \oplus t_1$ and $s = s_0 \oplus s_1$. The construction employs two simple ideas.

1. In the 2-party case, additive secret sharing satisfies the following weak homomorphism: If G is a PRG, then $G([s]) = (G(s_0), G(s_1))$ extends shares of the 0-string $s = 0$ into shares of a longer 0-string $S = 0$, and shares of a random seed s into shares of a longer (pseudo-)random string S , where S is pseudo-random even given one share of s .
2. Additive secret sharing is additively homomorphic: given shares $[s], [t]$ of a string s and a bit t , and a public correction word CW , one can locally compute shares of $[s \oplus (t \cdot CW)]$. We view this as a *conditional correction* of the secret s by CW conditioned on $t = 1$.

To maintain the above invariant along the evaluation path, we use the two types of homomorphism as follows. Suppose that the labels of the i -th node v_i on the evaluation path are $[s], [t]$. To compute the labels of the $(i + 1)$ -th node, the parties start by locally computing $[S] = G([s])$ for a PRG $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda+2}$, parsing $[S]$ as $[s^L, t^L, s^R, t^R]$. The first two values correspond to labels of the left child and the last two values correspond to labels of the right child.

To maintain the invariant, the keys will include a correction word CW for each level i . As discussed above, we only need to consider the case where v_i is on the special path. By the invariant we have $t = 1$, in which case the correction will be applied. Suppose without loss of generality that $\alpha_i = 1$. This means that the left child of v_i is off the special path whereas the right child is on the special path. To ensure that the invariant is maintained, we can include in both keys the correction $CW^{(i)} = (s^L, t^L, s^R \oplus s', t^R \oplus 1)$ for a random seed s' . Indeed, this ensures that after the correction is applied, the labels of the left and right child are $[0], [0]$ and $[s'], [1]$ as required. But since we do not need to consider the value of s' , except for making it pseudo-random, we can instead use the correction $CW^{(i)} = (s^L, t^L, s^L, t^R \oplus 1)$ that can be described using $\lambda + 2$ bits. This corresponds to $s' = s^L \oplus s^R$. The n correction values $CW^{(i)}$ are computed by Gen^\bullet from the root labels by applying the above iterative computation along the special path, and are included in both keys.

Finally, assuming that $\beta = 1$, the output of Eval^\bullet is just the shares $[t]$ of the leaf corresponding to x . A different value of β (from an arbitrary Abelian group) can be handled via an additional correction $CW^{(n+1)}$.

3.2 Overview: δ -HSS for Branching Programs from DDH

We next give a simplified overview of the HSS construction from [8], following exposition from [7]. Cast into the framework of Definition 4, this yields an additive public-key $(*, 2)$ - δ -HSS for the class of branching programs under the DDH assumption.

For simplicity of notation (and for greater efficiency), we assume circular security of ElGamal encryption. This assumption can be replaced by standard DDH by replacing ElGamal encryption with the circular secure public-key encryption scheme of Boneh, Halevi, Hamburg, and Ostrovsky [5], as shown in [8].

RMS Programs

The construction of [8] supports homomorphic evaluation of straight-line programs of the following form over inputs $w_i \in \mathbb{Z}$, provided that all intermediate computation values in \mathbb{Z} remain “small,” bounded by a parameter M (where the required runtime grows with this size bound).

Definition 6 (RMS programs). *The class of Restricted Multiplication Straight-line (RMS) programs consists of a magnitude bound 1^M and an arbitrary sequence of the four following instructions, each with a unique identifier id:*

- Load an input into memory: $(\text{id}, \hat{y}_j \leftarrow \hat{w}_i)$.
- Add values in memory: $(\text{id}, \hat{y}_k \leftarrow \hat{y}_i + \hat{y}_j)$.
- Multiply value in memory by an input value: $(\text{id}, \hat{y}_k \leftarrow \hat{w}_i \cdot \hat{y}_j)$.
- Output value from memory, as element of \mathbb{Z}_β : $(\text{id}, \beta, \hat{O}_j \leftarrow \hat{y}_i)$.

If at any step of execution the size of a memory value exceeds the bound M , the output of the program on the corresponding input is defined to be \perp . We define the size of an RMS program P as the number of its instructions.

In particular, RMS programs allow only multiplication of a memory value with an *input* (not another memory value). RMS programs with $M = 2$ are powerful enough to efficiently simulate boolean formulas, logarithmic-depth boolean circuits, and deterministic branching programs (capturing logarithmic-space computations). For concrete efficiency purposes, their ability to perform arithmetic computations on larger inputs can also be useful.

Encoding \mathbb{Z}_q Elements. Let \mathbb{H} be a prime-order group, with a subgroup \mathbb{G} of prime order q (the DDH group). Let g denote a generator of \mathbb{G} . For any $x \in \mathbb{Z}_q$, consider the following 3 types of two-party encodings:

LEVEL 1: “Encryption.” For $x \in \mathbb{Z}_q$, we let $[x]$ denote g^x , and $\llbracket x \rrbracket_c$ denote $([r], [r \cdot c + x])$ for a uniformly random $r \in \mathbb{Z}_q$, which corresponds to an ElGamal encryption of x with a secret key $c \in \mathbb{Z}_q$. (With short-exponent ElGamal, c is a 160-bit integer.) We assume that c is represented in base B ($B = 2$ by default) as a sequence of s digits $(c_i)_{1 \leq i \leq s}$. We let $\lllbracket x \rrlbracket_c$ denote $(\llbracket x \rrbracket_c, (\llbracket x \cdot c_i \rrbracket_c)_{1 \leq i \leq s})$. All level-1 encodings are known to both parties.

LEVEL 2: “Additive shares.” Let $\langle x \rangle$ denote a pair of shares $x_0, x_1 \in \mathbb{Z}_q$ such that $x_0 = x_1 + x$, where each share is held by a different party. We let $\langle\langle x \rangle\rangle_c$ denote $(\langle x \rangle, \langle x \cdot c \rangle) \in (\mathbb{Z}_q^2)^2$, namely each party holds one share of $\langle x \rangle$ and one share of $\langle x \cdot c \rangle$. Note that both types of encodings are additively homomorphic over \mathbb{Z}_q , namely given encodings of x and x' the parties can locally compute a valid encoding of $x + x'$.

LEVEL 3: “Multiplicative shares.” Let $\{x\}$ denote a pair of shares $x_0, x_1 \in \mathbb{G}$ such that the difference between their discrete logarithms is x . That is, $x_0 = x_1 \cdot g^x$.

Operations on Encodings

We manipulate the above encodings via the following two types of operations, performed locally by the two parties:

1. $\text{Pair}(\llbracket x \rrbracket_c, \langle\langle y \rangle\rangle_c) \mapsto \{xy\}$. This pairing operation exploits the fact that $[a]$ and $\langle b \rangle$ can be locally converted to $\{ab\}$ via exponentiation.
2. $\text{Convert}(\{z\}, \delta) \mapsto \langle z \rangle$, with failure bound δ . The implementation of Convert is also given an upper bound M on the “payload” z ($M = 1$ by default), and its expected running time grows linearly with M/δ . We omit M from the following notation.

The **Convert** algorithm works as follows. Each party, on input $h \in \mathbb{G}$, outputs the minimal integer $i \geq 0$ such that $h \cdot g^i$ is “distinguished,” where roughly a δ -fraction of the group elements are distinguished. Distinguished elements were picked in [8] by applying a pseudo-random function to the description of the group element. An optimized conversion procedure from [10] (using special “conversion-friendly” choices of $\mathbb{G} \subset \mathbb{Z}_p^*$ and $g = 2$) applies the heuristic of defining a group element to be distinguished if its bit-representation starts with $d \approx \log_2(M/\delta)$ leading 0’s; this was further optimized by considering instead the $(d + 1)$ -bit string $1||0^d$ in [7]. Note that this heuristic only affects the running time and not security, and thus it can be validated empirically. Correctness of **Convert** holds if no group element *between* the two shares $\{z\} \in \mathbb{G}^2$ is distinguished.

Finally, **Convert** can signal that there is a potential failure if there is a distinguished point in the “danger zone.” Namely, Party $b = 0$ (resp., $b = 1$) raises a potential error flag \perp if $h \cdot g^{-i}$ (resp., $h \cdot g^{i-1}$) is distinguished for some $i = 1, \dots, M$.

Note that we used the notation M both for the payload upper bound in **Convert** and for the bound on the memory values in the definition of RMS programs (Definition 6). In the default case of RMS program evaluation using base 2 for the secret key c in level 1 encodings, both values are indeed the same. (However, when using larger basis, they can differ in parts of the computation, and a more careful analysis can improve error bound guarantees.)

Let **PairConv** be an algorithm that sequentially executes the two operations **Pair** and **Convert** above: $\text{PairConv}(\llbracket x \rrbracket_c, \langle\langle y \rangle\rangle_c, \delta) \mapsto \langle xy \rangle$, with error δ . We denote by **Mult** the following algorithm:

- **Functionality:** $\text{Mult}(\llbracket\llbracket x \rrbracket\rrbracket_c, \langle\langle y \rangle\rangle_c, \delta) \mapsto \langle\langle xy \rangle\rangle_c$
 - Parse $\llbracket\llbracket x \rrbracket\rrbracket_c$ as $(\llbracket x \rrbracket_c, (\llbracket x \cdot c_i \rrbracket_c)_{1 \leq i \leq s})$.
 - Let $\langle xy \rangle \leftarrow \text{PairConv}(\llbracket x \rrbracket, \langle\langle y \rangle\rangle_c, \delta')$ for $\delta' = \delta/(s + 1)$.
 - For $i = 1$ to s , let $\langle xy \cdot c_i \rangle \leftarrow \text{PairConv}(\llbracket xc_i \rrbracket_c, \langle\langle y \rangle\rangle_c, \delta')$.
 - Let $\langle xy \cdot c \rangle = \sum_{i=1}^s B^{i-1} \langle xy \cdot c_i \rangle$.
 - Return $(\langle xy \rangle, \langle xy \cdot c \rangle)$.

HSS for RMS Programs

Given the above operations, an additive δ -HSS for RMS programs is obtained as follows. This can be cast as HSS in Definition 4 with a key generation setup.

- **KEY GENERATION:** $\text{Gen}(1^\lambda)$ picks a group \mathbb{G} of order q with λ bits of security, generator g , and secret ElGamal key $c \in \mathbb{Z}_q$. It outputs $\text{pk} = (\mathbb{G}, g, h, \llbracket c_i \rrbracket_c)_{1 \leq i \leq s}$, where $h = g^c$, and $(\text{ek}_0, \text{ek}_1) \leftarrow \langle c \rangle$, a random additive sharing of c .
- **SHARE:** $\text{Share}(\text{pk}, x)$ uses the homomorphism of ElGamal to compute and output $\llbracket x \rrbracket_c$.
- **RMS PROGRAM EVALUATION:** For an RMS program P of multiplicative size S , the algorithm $\text{Eval}(b, \text{ek}_b, (\text{ct}_1, \dots, \text{ct}_n), P, \delta, \beta)$ processes the instructions

of P , sorted according to id , as follows. We describe the algorithm for both parties b jointly, maintaining the invariant that whenever a memory variable \hat{y} is assigned a value y , the parties hold level-2 shares $Y = \langle\langle y \rangle\rangle_c$.

- $\hat{y}_j \leftarrow \hat{x}_i$: Let $Y_j \leftarrow \text{Mult}(\llbracket x_i \rrbracket_c, \langle\langle 1 \rangle\rangle_c, \delta/S)$, where $\langle\langle 1 \rangle\rangle_c$ is locally computed from $(\text{ek}_0, \text{ek}_1)$ using $\langle 1 \rangle = (1, 0)$.
- $\hat{y}_k \leftarrow \hat{y}_i + \hat{y}_j$: Let $Y_k \leftarrow Y_i + Y_j$.
- $\hat{y}_k \leftarrow \hat{x}_i \cdot \hat{y}_j$: Let $Y_k \leftarrow \text{Mult}(\llbracket x_i \rrbracket_c, Y_j, \delta/S)$.
- $(\beta, \hat{O}_j \leftarrow \hat{y}_i)$: Parse Y_i as $(\langle y_i \rangle, \langle y_i \cdot c \rangle)$ and output $O_j = \langle y_i \rangle + (r, r) \bmod \beta$ for a fresh (pseudo-)random $r \in \mathbb{Z}_q$.

The confidence flag is \perp if any of the invocations of `Convert` raises a potential error flag, otherwise it is \top .

The pseudorandomness required for generating the outputs and for `Convert` is obtained by using a common pseudorandom function key that is (implicitly) given as part of each ek_b , and using a unique nonce as an input to ensure that different invocations of `Eval` are indistinguishable from being independent.

A single-input (“secret-key”) HSS variant is simpler in two ways. First, `Share` can directly run `Gen` and generate $\llbracket x \rrbracket_c$ from the secret key c . Second, an input loading instruction $\hat{y}_j \leftarrow \hat{x}_i$ can be processed directly, without invoking `Mult`, by letting `Share` compute $Y_j \leftarrow \langle\langle x_i \rangle\rangle_c$ and distribute Y_j as shares to the two parties.

Performance. The cost of each RMS multiplication or input loading is dominated by $s + 1$ invocations of `PairConv`, where each invocation consists of `Pair` and `Convert`. The cost of `Pair` is dominated by one group exponentiation (with roughly 200-bit exponent in [7]). The basis of the exponent depends only on the key and the input, which allows for optimized fixed-basis exponentiations when the same input is involved in many RMS multiplications. When the RMS multiplications apply to 0/1 values (this is the case when evaluating branching programs), the cost of `Convert` is linear in BS/δ , where the B factor comes from the fact that the payload z of `Convert` is bounded by the size of the basis. When δ is sufficiently small, the overall cost is dominated by the $O(BS^2s/\delta)$ “conversion” steps, where each step consists of multiplying by g and testing whether the result is a distinguished group element.

4 Applications and Implications

In this section, we turn to implications of FSS and HSS constructions. We begin by describing what is known about the relation of FSS/HSS to other primitives, and then address applications of both “low-end” and “high-end” construction regimes.

4.1 Relation to Other Primitives

Below are the primary known theoretical implications of FSS/HSS primitives.

One-way functions. FSS for any “sufficiently rich” function class \mathcal{F} (e.g., point functions) necessitates the existence of OWF [28]. Further, in such an FSS, each

output share f_i viewed as a function on its own must define a pseudorandom function [6]. Note that this is not a-priori clear from the security definition, which only requires that the shares hide f .

(Amortized) Low-communication secure computation. It was shown in [6] that FSS for a function class \mathcal{F} strictly containing the decryption circuit for a secure symmetric-key encryption scheme implies amortized low-communication protocols for secure two-party computation of a related function class, relying on a reusable source of correlated randomness (that can be realized via one-time offline preprocessing). Given HSS for \mathcal{F} , the same result holds without needing to amortize over the preprocessing.⁶

At the time of this result, all known approaches for obtaining such protocols relied on fully homomorphic encryption or related primitives, and as such this was viewed as a “barrier” against achieving such FSS without FHE. In an interesting twist, this was reversed by the work of [8], which succeeded in constructing a form of HSS for NC^1 (and thus succinct secure computation) from DDH.

However, the “barrier” still seems legitimate as evidence against the possibility of constructing general FSS/HSS (or even classes such as NC^1 or possibly AC^0) from weak cryptographic assumptions such as the existence of one-way functions or oblivious transfer.

Non-interactive key exchange (NIKE) & 2-message oblivious transfer (OT). The power of additive *multi-input* HSS (where inputs from different parties can be homomorphically computed on together; c.f. Definition 4) seems to be much greater than its single-input counterpart. Whereas constructions for single-input HSS exist for some function classes from OWF, to date *all* constructions of multi-input HSS rely on a select list of heavily structured assumptions: DDH, LWE, and obfuscation [8, 23].

It appears this is in some sense inherent: As shown in [11], even a minimal version of 2-party, 2-server additive HSS for the AND of two input bits implies the existence of non-interactive key exchange (NIKE) [21], a well-studied cryptographic notion whose known constructions are similarly limited to select structured assumptions. NIKE is black-box separated from one-way functions and highly unlikely to be implied by generic public-key encryption or oblivious transfer.

On the other hand, this same type of $(2, 2)$ -additive-HSS for AND is unlikely to be implied *by* NIKE, as the primitive additionally implies the existence of 2-message oblivious transfer (OT) [8], unknown to follow from NIKE alone. Further connections from HSS to 2-round secure computation have been demonstrated in [10, 11].

Worst-case to average-case reductions. A different type of implication of HSS is in obtaining worst-case to average-case reductions in P . Roughly speaking, the HSS evaluation function Eval for homomorphically evaluating a function

⁶ Recall in HSS the secret share size scales with input size and not function description size.

F defines a new function F' such that computing F on any given input x can be reduced to computing F' on two or more inputs that are individually pseudo-random (corresponding to the HSS secret shares of x). A similar application was pointed out in [17] using fully homomorphic encryption (FHE) (and a significantly weaker version in [28] using DPF). Compared to the FHE-based reductions, the use of HSS has the advantages of making only a constant number of queries to a *Boolean* function F' (as small as 2), and minimizing the complexity of recovering the output from the answers to the queries. The latter can lead to efficiency advantages in the context of applications (including the settings of fine-grained average-case hardness and verifiable computation; see [11]). It also gives rise to worst-case to average-case reductions under assumptions that are not known to imply FHE, such as the DDH assumption.

4.2 Applications in the One-Way Function Regime

FSS in the “low-end” regime has interesting applications to efficient private manipulation of remotely held databases, extending the notions of Private Information Retrieval (PIR) [16] and Private Information Storage (PIS) [35] to more expressive instruction sets. Recently, FSS has also been shown to yield concrete efficiency improvements in secure 2-party computation protocols for programs with data-dependent memory accesses. We describe these in greater detail below.

Multi-server PIR and secure keyword search. Suppose that each of m servers holds a database D of keywords $w_j \in \{0, 1\}^n$. A client wants to count the number of occurrences of a given keyword w without revealing w to any strict subset of the servers. Letting $\mathbb{G} = \mathbb{Z}_{m+1}$ and $f = f_{w,1}$ (the point function evaluating to 1 on target value w), the client can split f into m additive shares and send to server i the key k_i describing f_i . Server i computes and sends back to the client $\sum_{w_j \in D} f_i(w_j)$. The client can then find the number of matches by adding the m group elements received from the servers. Standard PIR corresponds to the same framework with point function $f_{i,1}$ for target data index i . In this application, FSS for other classes \mathcal{F} can be used to accommodate richer types of search queries, such as counting the number of keywords that lie in an interval, satisfy a fuzzy match criterion, etc. We note that by using standard randomized sketching techniques, one can obtain similar solutions that do not only count the number of matches but also return the payloads associated with a bounded number of matches (see, e.g., [36]).

Splinter [40]. In this fashion, FSS for point functions and intervals are the core of the system Splinter [40] of Wang *et al.*, serving private search queries on a Yelp clone of restaurant reviews, airline ticket search, and map routing. On top of the functionalities offered directly by the FSS, the system supports more expressive queries, such as MAX/MIN and TOP- k , by manipulating the database on the server side such that a point function/interval search on the modified database answers the desired query. (Here the type of query is revealed, but the search parameters are hidden.) Splinter reports end-to-end latencies below 1.6 s for

realistic workloads, including search within a Yelp-like database comparable to 40 cities, and routing within real traffic-map data for New York City.

Incremental secret sharing. Suppose that we want to collect statistics about web usage of mobile devices without compromising the privacy of individual users, and while allowing fast collection of real-time aggregate usage data. A natural solution is to maintain a large secret-shared array of group elements between m servers, where each entry in the array is initialized to 0 and is incremented whenever the corresponding web site is visited. A client who visits URL u can now secret-share the point function $f = f_{u,1}$, and each server i updates its shared entry of each URL u_j by locally adding $f_i(u_j)$ to this share. The end result is that only position u_j in the shared array is incremented, while no collusions involving strict subsets of servers learn which entry was incremented. Here too, applying general FSS can allow for more general “attribute-based” writing patterns, such as secretly incrementing all entries whose public attributes satisfy some secret predicate. The above incremental secret sharing primitive can be used to obtain low-communication solutions to the problem of private information storage [35], the “writing” analogue of PIR.

Riposte [18]. FSS for point functions on a 2^{20} -entry database are used in this way in the anonymous broadcast system Riposte of Corrigan-Gibbs *et al.* [18]. Roughly, in the system each user splits his message `msg` as a point function $f_{r,\text{msg}}$ for a random position index $r \in [2^{20}]$. Shares of such functions across many users are combined additively by each server, and ultimately the *aggregate* is revealed. FSS security guarantees that the link from each individual user to his contributed message remains hidden.

Protecting against malicious clients. In some applications, malicious clients may have incentive to submit bogus FSS shares to the servers, corresponding to illegal manipulations of the database. This can have particularly adverse effects in writing applications, e.g., casting a “heavy” vote in a private poll, or destroying the current set of anonymous broadcast messages. Because of this, it is desirable to have efficient targeted protocols that enable a client to prove the validity of his request before it is implemented, via minimal interaction between the client and servers. Such protocols have been designed for certain forms of DPFs and related settings in [9, 18].

Secure 2-party computation (2PC) of RAM programs. A standard challenge in designing secure computation protocols is efficiently supporting *data-dependent* memory accesses, without leaking information on which items were accessed (and in turn on secret input values). Since the work of [35], this is typically addressed using techniques of *Oblivious RAM* (ORAM) [31] to transform a memory access to a secret index i from data size N into a sequence of $\text{polylog}(N)$ memory accesses whose indices appear independent of i . Indeed, a line of works in the past years have implemented and optimized systems for ORAM in secure computation.

Floram [39]. In a surprising recent development, Doerner and Shelat [39] demonstrated an *FSS-based* 2PC system that—despite its inherent poor $O(N)$

asymptotic computation per private access of each secret index i (instead of $\text{polylog}(N)$)—concretely outperforms current ORAM-based solutions.

In their construction, similar to use of ORAM in 2PC, the two parties in the secure computation act as the two servers in the FSS scheme, and an underlying (circuit-based) secure computation between the parties emulates the role of the client. The core savings of their approach is that, while overall computation is high, the emulation of “client” operations in the FSS requires a very small secure computation in comparison to prior ORAM designs (up to one hundred times smaller for the memory sizes they explore). Their implemented 2PC system *Floram* [39] (“**FSS Linear ORAM**”) outperforms the fastest previously known ORAM implementations, *Circuit ORAM* [41] and *Square-root ORAM* [43], for datasets that are 32 KiB or larger, and outperforms prior work on applications such as secure stable matching [24] or binary search [32] by factors of two to ten.

4.3 Applications in the Public-Key Regime

In the “high-end” regime, HSS can serve as a competitive alternative to FHE in certain application settings. Fully homomorphic encryption (FHE) [26, 37] is commonly viewed as a “dream tool” in cryptography, enabling one to perform arbitrary computations on encrypted inputs. For example, in the context of secure multiparty computation (MPC) [3, 13, 30, 42], FHE can be used to minimize the communication complexity and the round complexity, and shift the bulk of the computational work to any subset of the participants. However, despite exciting progress in the past years, even the most recent implementations of FHE [14, 25, 33] are still quite slow and require large ciphertexts and keys. This is due in part to the limited set of assumptions on which FHE constructions can be based [12, 22, 27], which are all related to lattices and are therefore susceptible to lattice reduction attacks. As a result, it is arguably hard to find realistic application scenarios in which current FHE implementations outperform optimized versions of classical secure computation techniques (such as garbled circuits) when taking both communication and computation costs into account.

A main motivating observation is that unlike standard FHE, HSS can be useful even for *small computations* that involve short inputs, and even in application scenarios in which competing approaches based on traditional secure computation techniques do not apply at all.

Advantages of HSS. As with FHE, HSS enables secure computation protocols that simultaneously offer a minimal amount of interaction and collusion resistance. However, the *optimal output compactness* of HSS makes it the only available option for applications that involve computing long outputs (or many short outputs) from short secret inputs (possibly along with public inputs). More generally, this feature enables applications in which the communication and computation costs of output reconstruction need to be minimized, e.g., for the purpose of reducing power consumption. For instance, a mobile client may wish to get quickly notified about live news items that satisfy certain secret search criteria, receiving a fast real-time feed that reveals only pointers to matching items.

Further advantages of group-based HSS over existing FHE implementations include smaller keys and ciphertexts and a lower startup cost.

HSS Applications

Applications of HSS include small instances of general secure multiparty computation, as well as distributed variants of private information retrieval, functional encryption, and broadcast encryption. Exploring concrete such applications (and optimizing the DDH-based δ -HSS construction) is the primary focus of [7].

Secure MPC with minimal interaction. Using multi-input HSS, a set of clients can outsource a secure computation to two non-colluding servers by using the following minimal interaction pattern: each client independently sends a single message to the servers (based on its own input and the public key), and then each server sends a single message to each client. Alternatively, servers can just publish shares of the output if the output is to be made public. The resulting protocol is resilient to any (semi-honest) collusion between one server and a subset of the clients, and minimizes the amount of work performed by the clients. It is particularly attractive in the case where many “simple” computations are performed on the same inputs. In this case, each additional instance of secure computation involves just local computation by the servers, followed by a minimal amount of communication and work by the clients.

Secure data access. HSS yields several different applications in the context of secure access to distributed data. For example, HSS can be used to construct a 2-server variant of attribute based encryption, in which each client can access an encrypted file only if its (public or encrypted) attributes satisfy an encrypted policy set up by the data owner. Other sample applications include 2-server private RSS feeds, in which clients can receive succinct notifications about new data that satisfies their encrypted matching criteria, and 2-server PIR schemes with general boolean queries. These applications benefit from the optimal output compactness feature of HSS discussed above, minimizing the communication from servers to clients and the computation required for reconstructing the output.

Unlike competing solutions based on classical secure computation techniques, HSS-based solutions only involve minimal interaction between clients and servers and no direct interaction between servers. In fact, for the RSS feed and PIR applications, the client is free to choose an arbitrary pair of servers who have access to the data being privately searched. These servers do not need to be aware of each other’s identity, and do not even need to know they are participating in an HSS-based cryptographic protocol: each server can simply run the code provided by the client on the (relevant portion of) the data, and return the output directly to the client.

Correlated randomness generation. An interesting application scenario is where the target output itself *is* an additive secret sharing. HSS provides a method for non-interactively generating sources of *correlated randomness* that can be used to speed up classical protocols for secure two-party computation.

Concretely, following a setup phase, in which the parties exchange HSS shares of random inputs, the parties can *locally* expand these shares (without any communication) into useful forms of correlated randomness. The non-interactive nature of the correlated randomness generation is useful for hiding the identities of the parties who intend to perform secure computation (e.g., against network traffic analysis), as well as the time and the size of the computation being performed.

Useful correlations considered in [7] include bilinear correlations (which capture “Beaver triples” as a special case) and truth-table correlations. The work of [7] also proposes further compression of communication in the setup phase by using homomorphic evaluation of local PRGs, and present different approaches for improving its asymptotic computational complexity. However, this PRG-based compression is still in the theoretical regime (too slow to be realized with good concrete running time using the current implementation of group-based HSS).

5 Future Directions

The study of FSS/HSS is a rapidly expanding new field of research, which continues to surprise and reveal even further layers of mystery. This survey is by no means a comprehensive coverage of all that is known, but rather seeks to facilitate future study by providing a semi-centralized resource with helpful pointers.

We close with a selection of open problems, as well as an excitement (on behalf of the author) for what is yet to come.

5.1 Open Problems

- Improved FSS from OWF
 - Improved DPF efficiency and/or lower bounds?
 - OWF-based FSS for CNF/DNF formulas? Better FSS for decision trees?
 - 3-server DPF with better than $2^{n/2}$ key size?
 - Separations between OWF and FSS for new function classes \mathcal{F} ?
- Improved HSS from DDH
 - Better error-to-computation tradeoff (in share conversion step of Eval)?
 - DDH-based error-free HSS for branching programs?
 - DDH-based δ -HSS for circuits (or anything beyond branching programs)?
 - DDH-based HSS for >2 servers?
- New types of constructions
 - FSS/HSS from new assumptions?
 - HSS from LWE without going through FHE?
- Applications & Implementations
 - Better optimizations for implementation?
 - Implementation of “high-end” HSS-based applications?
 - New application settings of FSS & HSS?

Acknowledgements. Tremendous thanks to my FSS/HSS partners in crime, Niv Gilboa and Yuval Ishai, and to additional coauthors on the presented works: Geoffroy Couteau, Huijia Rachel Lin, Michele Orrù, and Stefano Tessaro.

References

1. Beimel, A., Burmester, M., Desmedt, Y., Kushilevitz, E.: Computing functions of a shared secret. *SIAM J. Discrete Math.* **13**(3), 324–345 (2000)
2. Beimel, A., Ishai, Y., Kushilevitz, E., Orlov, I.: Share conversion and private information retrieval. In: *CCC 2012*, pp. 258–268 (2012)
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: *STOC*, pp. 1–10 (1988)
4. Benaloh, J.C.: Secret sharing homomorphisms: keeping shares of a secret secret (extended abstract). In: Odlyzko, A.M. (ed.) *CRYPTO 1986*. LNCS, vol. 263, pp. 251–260. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_19
5. Boneh, D., Halevi, S., Hamburg, M., Ostrovsky, R.: Circular-secure encryption from decision Diffie-Hellman. In: Wagner, D. (ed.) *CRYPTO 2008*. LNCS, vol. 5157, pp. 108–125. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_7
6. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) *EUROCRYPT 2015*. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
7. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: optimizations and applications. In: *ACM SIGSAC CCS*, pp. 2105–2122 (2017)
8. 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
9. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: improvements and extensions. In: *ACM SIGSAC CCS*, pp. 1292–1303 (2016)
10. 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
11. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: *ITCS* (2017)
12. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.* **43**(2), 831–871 (2014)
13. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: *STOC*, pp. 11–19 (1988)
14. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) *ASIACRYPT 2016*. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_1
15. Chor, B., Gilboa, N.: Computationally private information retrieval (extended abstract). In: *Proceedings of 29th Annual ACM Symposium on the Theory of Computing*, pp. 304–313 (1997)
16. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. *J. ACM* **45**(6), 965–981 (1998)
17. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Rabin, T. (ed.) *CRYPTO 2010*. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_26

18. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: An anonymous messaging system handling millions of users. In: IEEE Symposium on Security and Privacy, SP, pp. 321–338 (2015)
19. De Santis, A., Desmedt, Y., Frankel, Y., Yung, M.: How to share a function securely. In: Proceedings of 26th Annual ACM Symposium on Theory of Computing, pp. 522–533 (1994)
20. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, New York (1990). <https://doi.org/10.1007/0-387-34805-0-28>
21. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theor. **22**(6), 644–654 (1976)
22. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2
23. 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
24. Doerner, J., Evans, D., Shelat, A.: Secure stable matching at scale. In: ACM SIGSAC CCS, pp. 1602–1613 (2016)
25. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_24
26. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178 (2009)
27. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5
28. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
29. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986)
30. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229 (1987)
31. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)
32. Gordon, S.D., Katz, J., Kolesnikov, V., Krell, F., Malkin, T., Raykova, M., Vahlis, Y.: Secure two-party computation in sublinear (amortized) time. In: ACM CCS, pp. 513–524 (2012)
33. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 641–670. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_25
34. Kalyanasundaram, B., Schnitger, G.: The probabilistic communication complexity of set intersection. SIAM J. Discrete Math. **5**(4), 545–557 (1992)
35. Ostrovsky, R., Shoup, V.: Private information storage (extended abstract). In: ACM Symposium on the Theory of Computing, pp. 294–303 (1997)

36. Ostrovsky, R., Skeith III, W.E.: Private searching on streaming data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_14
37. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. In: Workshop on Foundations of Secure Computation, Georgia Institute of Technology, Atlanta, GA, pp. 169–179. Academic, New York (1978)
38. Shamir, A.: How to share a secret. *Commun. ACM* **22**(11), 612–613 (1979)
39. Doerner, J., Shelat, A.: Scaling ORAM for secure computation. In: ACM SIGSAC CCS, pp. 523–535 (2017)
40. Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: practical private queries on public data. In: 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI, pp. 299–313 (2017)
41. Wang, X., Chan, T.H., Shi, E.: Circuit ORAM: on tightness of the Goldreich-Ostrovsky lower bound. In: ACM SIGSAC CCS, pp. 850–861 (2015)
42. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS, pp. 162–167 (1986)
43. Zahur, S., Wang, X.S., Raykova, M., Gascón, A., Doerner, J., Evans, D., Katz, J.: Revisiting square-root ORAM: efficient random access in multi-party computation. In: IEEE Symposium on Security and Privacy, SP, pp. 218–234 (2016)

A Note on Ring-LWE Security in the Case of Fully Homomorphic Encryption

Guillaume Bonnoron¹ and Caroline Fontaine²(✉)

¹ Chair of Naval Cyber Defense – Ecole Navale, CC600,
29240 Brest Cedex 9, France

guillaume.bonnoron@ecole-navale.fr

² CNRS/Lab-STICC – IMT Atlantique, CS 83818, 29238 Brest Cedex 3, France
caroline.fontaine@imt-atlantique.fr

Abstract. Evaluating the practical security of Ring-LWE based cryptography has attracted lots of efforts recently. Indeed, some differences from the standard LWE problem enable new attacks. In this paper we discuss the security of Ring-LWE as found in Fully Homomorphic Encryption (FHE) schemes. These FHE schemes require parameters of very special shapes, that an attacker might use to its advantage. First we present the specificities of this case and recall state-of-the-art attacks, then we derive a new special-purpose attack. Our experiments show that this attack has unexpected performance and confirm that we need to study the security of special parameters sets carefully.

Keywords: Homomorphic encryption · Concrete security
Implementation · Ring-LWE

1 Introduction

The Learning With Errors over Rings (Ring-LWE) problem has been introduced by [LPR10] as a ring variant of the Learning With Errors due to Regev [Reg05]. Both problems enjoy security reductions to hard lattice problems (SIVP for LWE and SVP in ideal lattices for Ring-LWE), so they serve as hardness grounds for many cryptographic constructions, among others homomorphic encryption. See the survey from Peikert [Pei15] for an extensive retrospective. Today, the paramount question, that still stands in the way to practical use, concerns the security of concrete instances of these problems. Namely, how shall one choose parameters for these problems to meet a security level objective, say 80 bits of security?

Homomorphic encryption (HE) is a type of encryption that allows to compute over encrypted data. The result, once decrypted, equals that of the same computation done over the plain data. HE enables many new applications because one no longer needs to trust the computing entity, e.g. cloud service providers. Fully homomorphic encryption (FHE) was first achieved by the

Funded and supported by Ecole navale, IMT Atlantique, Thales and Naval Group.

© Springer International Publishing AG 2017

A. Patra and N. P. Smart (Eds.): INDOCRYPT 2017, LNCS 10698, pp. 27–43, 2017.

https://doi.org/10.1007/978-3-319-71667-1_2

ground-breaking works of Gentry [Gen09] and Aguilar et al. [AMGH10]. Since then, huge efforts have been spent and have led to numerous scheme proposals, e.g. [vDGHV10, BGV12, FV12, GSW13, DS15, DM15, CCGH16]. Some of the most efficient ones are now implemented, for early users: HELib [Hal], SEAL [DGBL+15], FV-NFLlib [Cry]. These efficient schemes are known secure under the assumption that Ring-LWE is intractable. For an application, the parameters need to be chosen to guarantee the expected security level. This choice of parameters is also constrained so that the scheme itself works.

Related work: This need for a better understanding of the Ring-LWE security, *in practice*, has already driven some studies. For example, Albrecht et al. [APS15] offer a complete overview of the known attacks against LWE. They also give estimates of their costs against some LWE-based cryptographic schemes, not only FHE. We recall them briefly below for our later discussion. Another more recent line of work has been developed against Ring-LWE specifically, taking advantage of the underlying ring structure, see [Pei16] for a summary and guidelines to draw immune parameters.

Today, the estimates we can use to choose parameters are from the works of Lindner and Peikert [LP11] or Van de Pol and Smart [vdPS13]. Also Albrecht et al. maintain an LWE security estimator based on models of state-of-the-art attacks¹. However none of them include special-purpose attacks for FHE settings.

Our work: It seems to us that a focus on Ring-LWE-based FHE scheme is needed. As we see later, in order to keep correctness in the homomorphic schemes, an application designer needs to pick very special parameters. It seemed unclear to us what concrete advantage an attacker might have in such cases. Our contribution aims at filling this gap. After reviewing state-of-the-art attacks, we derive a new one, specially designed for this case, and present experimental results.

Roadmap: Introducing notation and definitions in Sect. 2, we review the state-of-the-art of the attacks against Ring-LWE in Sect. 3. In Sect. 4 we present our new attack in details, its performance in Sect. 5 and draw some conclusions in Sect. 6.

2 Preliminaries

2.1 Notation

For a positive integer $q > 0$ we note \mathbb{Z}_q the set of elements $\{0, \dots, q - 1\}$.

For $n > 0$ integer, the matrix \mathbf{I}_n refers to the identity matrix of size n . Capital bold letters are used for matrices, e.g. \mathbf{B} , and small ones for (row) vectors, e.g. \mathbf{v} . We similarly write \mathbf{B} for a matrix or for the ordered family of (row) vectors $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n)$ using bold subscripts. For a vector \mathbf{v} , we refer to its components with italic subscripts v_i .

¹ <https://bitbucket.org/malb/lwe-estimator/overview>.

When dealing with a polynomial \mathbf{p} of degree $n - 1$, we use the coefficient embedding and assimilate it as a coordinate vector: $\mathbf{p} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n)$. We will work with polynomials in polynomial ring $R = \mathbb{Z}[x]/(f(x))$ for some $f \in \mathbb{Z}[x]$. We denote R_q the set of polynomials in R with coefficients in \mathbb{Z}_q .

2.2 Lattices

General definitions. In general, a lattice \mathcal{L} of dimension n is a discrete additive subgroup of \mathbb{R}^n . *Integer* lattices are discrete additive subgroups of \mathbb{Z}^n . In this paper we only work with the integer lattices and simply call them lattices.

Lattices (of size n) are usually represented by a basis \mathbf{B} , a set of n independent integer vectors $(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n)$ of size n whose integer linear combinations generate the lattice.

$$\mathcal{L}(\mathbf{B}) = \left\{ \sum_{i=1}^n v_i \mathbf{b}_i : v_i \in \mathbb{Z} \right\} = \{ \mathbf{B}^T \mathbf{v} : \mathbf{v} \in \mathbb{Z}^n \} = \mathbf{B} \mathbb{Z}^n$$

In our lattices, \mathbf{B} is always a square integer matrix, $\mathbf{B} \in \mathbb{Z}^{n \times n}$. For most of the discussion we restrict to this full rank definition and will make it explicit when working with greater generating families.

An invariant of a lattice is its determinant $\det(L)$. It is defined as the absolute value of the determinant of any of its bases.

$$\det(L) = |\det(\mathbf{B})|$$

Gram-Schmidt Orthogonalization (GSO). We refer several times to the GSO of a basis. This algorithm takes the matrix to orthogonalize and outputs the resulting matrix \mathbf{B}^* and a matrix μ (lower triangular with 1 on the diagonal) such that $\mathbf{B} = \mu \times \mathbf{B}^*$. We construct \mathbf{B}^* so that its vectors verify:

- $\mathbf{b}_1^* = \mathbf{b}_1$
- \mathbf{b}_i^* is the projection of \mathbf{b}_i orthogonally to the subspace generated by the $i - 1$ first vectors of \mathbf{B} .

It works in polynomial time in the size of the matrix.

q-ary lattices. When studying LWE and Ring-LWE problems, the lattices we mostly work with are called *q-ary*, because they are defined *modulo* some integer q (not necessarily prime). These lattices are defined as follows:

$$\begin{aligned} \mathcal{L}_q(\mathbf{B}) &= \left\{ \sum_{i=1}^n v_i \mathbf{b}_i \bmod q : v_i \in \mathbb{Z} \right\} \\ &= \{ \mathbf{B}^T \mathbf{v} \bmod q : \mathbf{v} \in \mathbb{Z}^n \} \end{aligned}$$

Hence, since we are working modulo q , we can equivalently consider that all the components of \mathbf{B} and \mathbf{v} are in \mathbb{Z}_q .

Lattice reduction. As we discuss below, working with lattices is better when we have *nice* bases. Therefore we have lattice reduction algorithms that, given a basis, make it nicer according to some criteria. There is a wealth of studies on this computational problem.

We usually consider the following criteria and say that a basis is:

- η -size-reduced if $\forall i < j, |(\mathbf{b}_j | \mathbf{b}_i^*)| \leq \eta \cdot \|\mathbf{b}_i^*\|^2$
- δ -LLL-reduced if it is size-reduced and $\forall i, \delta \|\mathbf{b}_i^*\|^2 \leq \left(\|\mathbf{b}_{i+1}^*\|^2 + \frac{(\mathbf{b}_{i+1} | \mathbf{b}_i^*)^2}{\|\mathbf{b}_i^*\|^2} \right)$
- β -BKZ-reduced if it is LLL-reduced and for all j , \mathbf{b}_j^* is the shortest vector of the sublattice spanned by $(\mathbf{b}_j, \dots, \mathbf{b}_k)$ with $k = \min(j + \beta - 1, n)$

We have algorithms to achieve such reductions:

- The celebrated LLL algorithm from Lenstra et al. [LLL82], running in polynomial time of the dimension and the size of the elements. Improvements have been proposed since then and are available in current implementations [NS05].
- The Blockwise Korkine-Zolotarev algorithm [SE94, CN11] achieves BKZ reduction. It makes a polynomial (in n) call to a SVP oracle in a sublattice of size β . It behaves roughly as a sub-exponential in the basis quality [GN08b] and uses LLL as a sub-routine.
- Slide Reduction [GN08a] is another block algorithm, in the spirit of BKZ, but simpler to express and analyze, whose performance approaches that of BKZ [MW15].

The quantity we usually use to measure the reduction quality, independently of the algorithm, is the *root Hermite factor* γ . It is defined as: $\|\mathbf{b}_1\| = \gamma^n \det(L)^{1/n}$ where \mathbf{b}_1 is the smallest vector of the basis we qualify. The higher the quality, the smaller it gets.

2.3 Learning with Errors and Ring Variant

We recall here the definitions of the Learning With Errors problem [Reg05] and the Ring-LWE variant [LPR10]. Both exist in a *search* version and a *decision* version.

Definition LWE. Let n, q be positive integers, χ a probability distribution on \mathbb{Z} of standard deviation σ and \mathbf{s} a secret random vector in \mathbb{Z}_q^n . We denote by $L_{\mathbf{s}, \chi}$ the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in \mathbb{Z}$ according to χ and considering it in \mathbb{Z}_q , and returning $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.

Decision-LWE is the problem of deciding whether given pairs (\mathbf{a}, c) are sampled according to $L_{\mathbf{s}, \chi}$ or the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Search-LWE is the problem of recovering \mathbf{s} from pairs (\mathbf{a}, c) sampled from $L_{\mathbf{s}, \chi}$.

Definition Ring-LWE. Let R be a ring of degree n over \mathbb{Z} (usually $R = \mathbb{Z}[x]/(f(x))$ for some cyclotomic polynomial $f(x)$). Let q be a positive integer, χ a probability distribution on R of standard deviation σ and \mathbf{s} a secret random element in R_q . We denote by $L_{\mathbf{s},\chi}$ the probability distribution on $R_q \times R_q$ obtained by choosing $\mathbf{a} \in R_q$ uniformly at random, choosing $\mathbf{e} \in R$ according to χ and considering it in R_q , and returning $(\mathbf{a}, \mathbf{c}) = (\mathbf{a}, [\mathbf{a} \cdot \mathbf{s} + \mathbf{e}]_q) \in R_q \times R_q$.

Decision-Ring-LWE is the problem of deciding whether given pairs (\mathbf{a}, \mathbf{c}) are sampled according to $L_{\mathbf{s},\chi}$ or the uniform distribution on $R_q \times R_q$.

Search-Ring-LWE is the problem to recovering \mathbf{s} from pairs (\mathbf{a}, \mathbf{c}) sampled from $L_{\mathbf{s},\chi}$.

The hardnesses of (Ring-)LWE problems depend on the three variables n , σ and q . The hardness reductions presented in the introductory papers stands when $\sigma > 2\sqrt{n}$. Besides that, we aim to establish a link between a choice of parameters and the provided security level, in the context of FHE.

2.4 Ring-LWE Based FHE Schemes

Since the works of Gentry [Gen09] and Aguilar et al. [AMGH10], lots of homomorphic encryption schemes have been proposed. They can be divided into two families: those based on integers and those based on lattices. LWE and Ring-LWE serve as building ground for the latter family. Generally, those based on Ring-LWE derive from an equivalent scheme on LWE and are more efficient in terms of space and/or time. The most common Ring-LWE based schemes are: [BGV12], [FV12] and SHIELD [KGV15].

For expository purpose, we recall here only elements of the [FV12] scheme. The discussion remains valid for all schemes as well. The interested reader should refer to the original papers for extensive details about the schemes. In our present study, we are interested in the elements that will be the attack target, namely the public key. In [FV12] the key generation process goes as follows:

1. **FV.ParameterChoice**(λ): choose (n, σ, q) to guarantee of level of security λ and set $R = \mathbb{Z}[x]/(\Phi_n(x))$
2. **FV.KeyGen**(n, σ, q): sample $\mathbf{s} \leftarrow R_2$, $\mathbf{a} \leftarrow R_q$, $\mathbf{e} \leftarrow \chi$ and output

$$\mathbf{sk} = \mathbf{s} \text{ and } \mathbf{pk} = ([-\mathbf{a} \cdot \mathbf{s} + \mathbf{e}]_q, \mathbf{a})$$

Security. Let aside the sign of the first element, the public key \mathbf{pk} is exactly a Ring-LWE pair as described above. The objective of the attacker is to solve *search-Ring-LWE* (i.e. find \mathbf{s}) when given access to this public key and the public parameters $(n, \sigma, q$ and $R)$.

Correctness. The condition for the scheme to be correct (i.e. decryption yields a result consistent with the computation) is [FV12, Eq. 6]:

$$4 \cdot \beta(\epsilon) \cdot \delta_R^{L_{\max}} \cdot (\delta_R + 1.25)^{L_{\max}+1} \cdot t^{L_{\max}-1} < \frac{q}{\sigma}$$

where

- L_{\max} is the maximum multiplicative depth before bootstrapping is needed.
- t is the size of the plaintext space (R_t).
- $\delta_R = \max\{\|\mathbf{a} \cdot \mathbf{b}\| / (\|\mathbf{a}\| \cdot \|\mathbf{b}\|) : \mathbf{a}, \mathbf{b} \in R\}$, e.g. $\delta_{\mathbb{Z}[x]/(x^n+1)} = n$.
- $\beta(\epsilon)$ is such that the error samples are bounded by $B = \beta(\epsilon) \cdot \sigma$ with probability $1 - \epsilon$, e.g. $\beta(2^{-64}) \approx 9.2$.

As a result of this constraint, we can see that q will have to be very large, in front of σ , to conserve correctness with interesting depth, say $L = 9$. To prevent q from being enormous, many authors tend to take σ very tiny, e.g. 3.2 or 8. However, this completely violates the bound $\sigma > 2\sqrt{n}$ required for the hardness reductions to hold [LPR10].

3 Existing Attacks

In January 2015, Albrecht et al. [APS15] aimed at assessing the concrete hardness of LWE and provided an excellent survey of the state-of-art approaches as of this date. These methods apply also to Ring-LWE. We recall here, from a high level perspective, the different families of methods, together with more recent attacks when such exist. The sections of their paper contains extensive details and are therefore mentioned here as reference for the interested reader. Then we briefly present the attacks targeting Ring-LWE specifically and summarized in the recent survey [Pei16].

Bruteforce on s . Attempting an exhaustive search is always possible, yet rarely efficient to solve the problem if the parameter choice is sound. For completeness they express the time complexity of such attack, in general [APS15, Sect. 5.1] and in the case where $\|\mathbf{s}\| \leq 1$ [APS15, Sect. 6.1]. There are better performing methods, as follow.

Distinction. In [APS15, Sect. 5.3], they present a way to distinguish LWE samples from uniformly random samples, as stated by the Decision-LWE problem. However this does not directly recover the secret.

Arora-Ge Attack [AG11]. A purely algebraic attack [APS15, Sects. 5.6 and 6.5], this one consists in constructing from LWE samples a polynomial whose root is the secret. It is particularly efficient in cases where $\|\mathbf{e}\|$ is very small, for example binary errors. For this case however, we are outside of the domain of validity of the hardness reduction from [Reg05] which requires $\sigma > 2\sqrt{n}$.

Blum-Kalai-Wasserman [BKW03]. This combinatorial method works like the Gauss elimination procedure [APS15, Sects. 5.2 and 6.4]. It requests many samples, searches collisions between parts of these samples and gradually creates linear combinations where more and more components equal 0. At the end, the combination allows to deduce the secret. This method has been improved a lot since its original presentation. To date, the best is from Guo et al. [GJS15]. Due to the number of samples required, we do not consider this attack further. Indeed we prefer to assume availability of only the public key and see what can be done.

Decoding. In LWE samples, Gaussian errors can be considered bounded, with high probability, by a sufficiently large bound. An approach is then to use decoding algorithms to remove the error. This addresses the LWE problem as a *Bounded-Distance Decoding (BDD)* instance in a specially crafted lattice. The BDD problem asks to recover the lattice point closest to a given one (not in the lattice), with the promise that the target point is close to the lattice [APS15, Sect. 5.4].

Decoding with Embedding. Another technique to cope with the difficulties of direct decoding is to embed the lattice into another. The latter has higher dimension and more properties that enable some optimizations. Two embeddings are presented: one in the general case [APS15, Sect. 5.5] by Kannan [Kan87] and one in the specific case of $\|\mathbf{s}\| \leq 1$ [APS15, Sect. 6.3] from [BG14].

Using the Ring Structure. Several recent works [EHL14, ELOS15, CLS15, CIV16] use the underlying ring structure to attack Ring-LWE. Peikert describe a unified framework that encompasses all these attacks and sort them into two classes: reduction to errorless LWE and reduction modulo an ideal for which decision-Ring-LWE is tractable [Pei16]. Consequently, we know some rings to be vulnerable and others to be immune.

4 Our New Attack

Out of these surveys, we conclude to keep the decoding attacks. Algebraic attacks are not usable within the bounds of hardness reductions [Reg05, LPR10] and combinatorial attacks require too many samples, we let them aside. In the case of [FV12], the secret is small, $\|\mathbf{s}\| \leq 1$, so when it comes to embedding, the solution [BG14] is best.

4.1 Bai-Galbraith Embedding Improved

We detail here a slightly improved version of this embedding.

Let $(\mathbf{A}, \mathbf{b} = \mathbf{A}^T \mathbf{s} + \mathbf{e} \text{ mod } q)$ be a LWE instance for some n, q and σ . As we derive the matrix \mathbf{A} from a single Ring-LWE sample (the key), we fix $m = n$, $\mathbf{A} \in \mathbb{Z}^{n \times n}$. Write $\mathbf{A}' = (\mathbf{A} | \mathbf{I}_n)$, being a $n \times 2n$ matrix. We have the following equality

$$\mathbf{b} = \mathbf{A}' \begin{pmatrix} \mathbf{s} \\ \mathbf{e} \end{pmatrix} \text{ mod } q$$

where $\begin{pmatrix} \mathbf{s} \\ \mathbf{e} \end{pmatrix}$ is a short vector with respect to q .

Clearly for $\mathbf{w} = \begin{pmatrix} \mathbf{0} \\ \mathbf{b} \end{pmatrix}$ we have $\mathbf{A}' \mathbf{w} = \mathbf{b}$. We then try to approximate this vector \mathbf{w} by a point \mathbf{v}_0 from the lattice \mathcal{L}' defined by

$$\mathcal{L}' = \{ \mathbf{v} \in \mathbb{Z}^{2n} : \mathbf{A}' \mathbf{v} = \mathbf{0} \text{ mod } q \}$$

For such a vector \mathbf{v}_0 , we have $\mathbf{v}_0 \approx \mathbf{w}$ so $\mathbf{w} - \mathbf{v}_0$ is a short vector (with respect to q) and

$$\begin{aligned} \mathbf{A}'(\mathbf{w} - \mathbf{v}_0) \bmod q &= \mathbf{A}'\mathbf{w} - \mathbf{A}'\mathbf{v}_0 \bmod q \\ &= \mathbf{b} - 0 \bmod q \\ &= \mathbf{b} \end{aligned}$$

Consequently we have $\mathbf{w} - \mathbf{v}_0 = \begin{pmatrix} \mathbf{s} \\ \mathbf{e} \end{pmatrix}$. Hence, finding \mathbf{v}_0 in \mathcal{L}' allows us to recover \mathbf{s} (and also \mathbf{e}). We can use BDD algorithms for this, if we have a basis \mathbf{B} for \mathcal{L}' . We obtain it by embedding as follows:

$$\mathbf{B}^T = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} \\ -\mathbf{A} & q\mathbf{I}_n \end{pmatrix} \in \mathbb{Z}^{2n \times 2n}$$

It can be verified that the columns of this matrix are linearly independent and each of them satisfies the definition of \mathcal{L}' , so \mathbf{B} is a basis of \mathcal{L}' .

This slightly differs from what Bai and Galbraith did. They introduced the matrix

$$\mathbf{M} = \begin{pmatrix} \mathbf{I}_n & \Big| & \\ -\mathbf{A} & \Big| & q\mathbf{I}_{2n} \end{pmatrix} \in \mathbb{Z}^{2n \times 3n}$$

and compute its column Hermite Normal Form to end up with a full rank matrix generating the lattice. Our trick avoids this HNF computation and yields a basis for \mathcal{L}' more efficiently².

After the embedding part, we leave untouched the rescaling operation from [BG14]. Since $\|\mathbf{s}\| \leq 1$ whereas the standard deviation for \mathbf{e} is σ (which is not small as discussed above), the components of the difference $\mathbf{w} - \mathbf{v}_0$ are unbalanced. Therefore we multiply the n first components of the basis vectors. It makes the lattice reduction easier, the difference more balanced and still short (with respect to q). Consequently the next stage of the attack works in the lattice whose basis is

$$\mathbf{B}^T = \begin{pmatrix} \sigma\mathbf{I}_n & \mathbf{0} \\ -\mathbf{A} & q\mathbf{I}_n \end{pmatrix} \in \mathbb{Z}^{2n \times 2n}$$

4.2 Lattice Reduction

Once the embedding and rescaling are done, the next step is to find the closest point \mathbf{v}_0 to \mathbf{w} in \mathcal{L}' . The existing algorithms for BDD require a somehow reduced basis of the lattice, otherwise they are completely impracticable. Since our basis \mathbf{B} in its present form verifies none of the criteria for reduction given at the beginning Sect. 2.2, we need to reduce it. We detail here the different options and our strategy.

Lattice reduction has been studied for long since it is very useful in cryptanalysis, integer programming, to cite only a few. The first and celebrated work of

² Computing the HNF of a matrix is not an intense computation, but can be avoided. See [SL96] for a complexity analysis.

Lenstra et al. [LLL82] presented a polynomial time algorithm for lattice reduction, whose output quality suffices for many applications. Since then, several blockwise algorithms have been introduced, the best performing are BKZ [CN11] and Slide Reduction [GN08a]. Both achieve better output quality than LLL but have worse time complexities. Since BDD becomes quicker with a better reduced basis, we need to find a trade-off between the time spent on the reduction step and on the BDD step, so that the overall attack time is minimized. Therefore, an attacker might attempt to use LLL first and if the BDD step takes too much time, try a stronger reduction algorithm. Same discussion apply when the algorithm is fixed, the attacker can tweak its parameters and achieve different output qualities at different computational costs.

The attacker may also look for specialised lattice reduction algorithm since the previous ones work for any lattices and do not take advantage of any structure in them. To the best of our knowledge, the only specialized algorithm that fits our case is the variant from Gama et al. [GHGN06]. The lattices it reduces are *symplectic*. Denoting $J_{2n} = \begin{pmatrix} \mathbf{0} & \mathbf{I}_n \\ -\mathbf{I}_n & \mathbf{0} \end{pmatrix} \in \mathbb{Z}^{2n \times 2n}$, a lattice with basis \mathbf{B} is symplectic if and only if $\mathbf{B}^T J_{2n} \mathbf{B} = J_{2n}$. The bases of our case are indeed symplectic, as can be verified. So this variant of LLL would be interesting to try. The authors reported a speed-up factor of nearly 10 when compared to reference implementation of classical LLL. However the code is not available, so we were not able to use it in our tests.

In the next section we present the details of our experiments on algorithms and parameters. Our conclusion is that for our case, the BDD step is successful even when we only perform an LLL reduction with quite weak parameters. This fact is absolutely no general conclusion. Here we are in a specific case, our basis \mathbf{B} has several properties (integer, upper triangular, blockwise upper triangular with scaled identity on the diagonal, etc.) and so do our parameters n , σ and q .

4.3 Pruned Enumeration for BDD

Finally, to find \mathbf{v}_0 in \mathcal{L}' close to \mathbf{w} with the reduced basis, we use Liu-Nguyen pruned enumeration [LN13]. To our knowledge pruned enumeration is the best performing algorithms in practice, see [HPS11] for a description of the other candidates.

This method of pruned enumeration is an adaptation to BDD of the extreme pruning technique introduced by Gama et al. [GNR10]. This algorithm enumerates lattice points that are close enough to the target, with a bound provided as a *pruning function*. Since the enumeration works by adding one component at a time to the current partial solution, we may define different bounds for each positions. Unlike previous work [Bab86, LP11], Liu-Nguyen pruned enumeration evaluates the heuristic on the *projections* of the current candidate and not on its *components*.

This concludes the algorithmic description of our attack. We turn now to implementation aspects and the results we get.

5 Implementation and Results

In this section we present the implementation details of our new attack, described in the previous section. Since we were able to use it successfully against many [FV12] keys, we thoroughly analyse its performance and give out some conclusions.

5.1 Implementation Details

In order to implement our attack, we use the following existing codes:

- Lepoint’s implementation [Lep14] of [FV12],
- Stehlé’s `fp111` [dt16] for lattice reduction algorithms,
- Shoup’s NTL [Sho15] for additional lattice operations.

Then, we implemented the embedding/rescaling, the pruned enumeration algorithm from Liu-Nguyen (following their pseudo-code [LN13]) and created the glue between the different libraries to lead the attack from beginning to end.

Lepoint’s implementation. In [LN14], in order to compare [FV12] and YASHE, Lepoint implemented the code needed to use both schemes and perform homomorphic operations. We use his constructor method with light modification. It allows us to create a public key $\mathbf{pk} = ([-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e})]_q, \mathbf{a})$, given n , σ and q . The attack aims to recover the secret when given the `FVKey` object as input.

From [FV12] to R -LWE lattice. From the \mathbf{a} in the public key, we construct a lattice basis \mathbf{A} so that $[\mathbf{a} \cdot \mathbf{s} + \mathbf{e}]_q$ (polynomial operation) equals $\mathbf{A}\mathbf{s} + \mathbf{e} \pmod{q}$ (matrix/vector operation). For instance, with $R = \mathbb{Z}[x]/(x^n + 1)$ we have:

$$\mathbf{A}^T = \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_n \\ -a_n & a_1 & a_2 & \cdots & a_{n-1} \\ -a_{n-1} & -a_n & a_1 & \cdots & a_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_2 & -a_3 & -a_4 & \cdots & a_1 \end{pmatrix}$$

Embedding and rescaling. Then we embed this matrix \mathbf{A} into \mathbf{B} following Sect. 4.1. Rescaling is also done here. \mathbf{B} is ready for the next stage.

Lattice reduction. For this step we rely on the different routines from `fp111` [dt16]. Both LLL and BKZ algorithms are available and can be tweaked conveniently. We detail below the settings we experimented.

BDD enumeration. Finally, we implemented the pruned enumeration algorithm from Liu-Nguyen to solve BDD in the reduced lattice \mathcal{L}' . Later, we discovered an undocumented implementation of enumeration in `fp111`, that we included in our experiments.

5.2 Attack Settings – Early Benchmarks

Before launching a full-scale attack, we run it on small cases. It allows us to explore the choices we have, for algorithms and their parameters, for the different stages.

Lattice Reduction Tweaking. State-of-the-art results [APS15] tell us that strong reductions like BKZ is needed to make enumeration possible. Yet after a few toy examples with BKZ as reduction algorithm, we realized that most of the attack time was spent in the lattice reduction stage. So we tried different settings for BKZ and LLL on cases where $n \leq 100$.

Beginning with BKZ, we could confirm [CN11], an intermediate blocksize $\beta = 20$ leads to moderate running times. Whereas smaller values like 5 or greater like 40 lead to prohibitive running times. Even with such blocksize, the attack time is not balanced between reduction and enumeration, so we rapidly shifted to study LLL.

LLL algorithm is governed by two parameters δ and η . We recall the size reduction condition and the Lovász condition from Sect. 2.2:

$$\begin{aligned} \forall i < j, \quad |(\mathbf{b}_j | \mathbf{b}_i^*)| &\leq \eta \cdot \|\mathbf{b}_i^*\|^2 && \text{with } 1/2 \leq \eta \leq \sqrt{\delta} \\ \forall i, \quad \delta \|\mathbf{b}_i^*\|^2 &\leq \left(\|\mathbf{b}_{i+1}^*\|^2 + \frac{(\mathbf{b}_{i+1} | \mathbf{b}_i^*)^2}{\|\mathbf{b}_i^*\|^2} \right) && \text{with } 1/4 \leq \delta \leq 1 \end{aligned}$$

The default values in `fp111` are $(\delta, \eta) = (0.99, 0.51)$.

Optimising δ . First, we tried to decrease δ to loosen the Lovász condition, however with $(0.75, 0.51)$ we gained only limited speedups and only for $n \geq 100$, and it had dramatic effect on the observed success rate of the overall attack. The enumeration stage fails due to a lattice basis of insufficient quality.

Optimising η . Then we restored δ and experimented with an increased $\eta \in [0.60, 0.98]$, to ease the size-reduction condition.

As we can see in Fig. 1, loosening the size reduction condition decreases the running time of LLL. We observe a greater gain (in time) between 0.60 and 0.75 than between 0.75 and 0.95. In addition, with $\eta = 0.95$ we get a less successful attack, whereas for η equal to 0.51 or 0.75 the observed success rates are similar.

Observed root Hermite factor. We report in Table 1 the quality we get with reduction parameters $\delta = 0.99$ and $\eta = 0.71$. As a reminder, the best proved γ for LLL is $\gamma = 1.0754$ and Gama and Nguyen observed $\gamma = 1.0219$ for random lattices of similar dimensions [GN08b]. In our case, we observe that LLL reduces lattices to a very good quality, even with weak reduction parameters δ and η .

Enumeration Behavior. For the enumeration step, we use the technique of pruned enumeration from [LN13]. Its only setting is the pruning function (R_1^2, \dots, R_{2n}^2) . As we know precisely the expected distance between \mathbf{w} and the

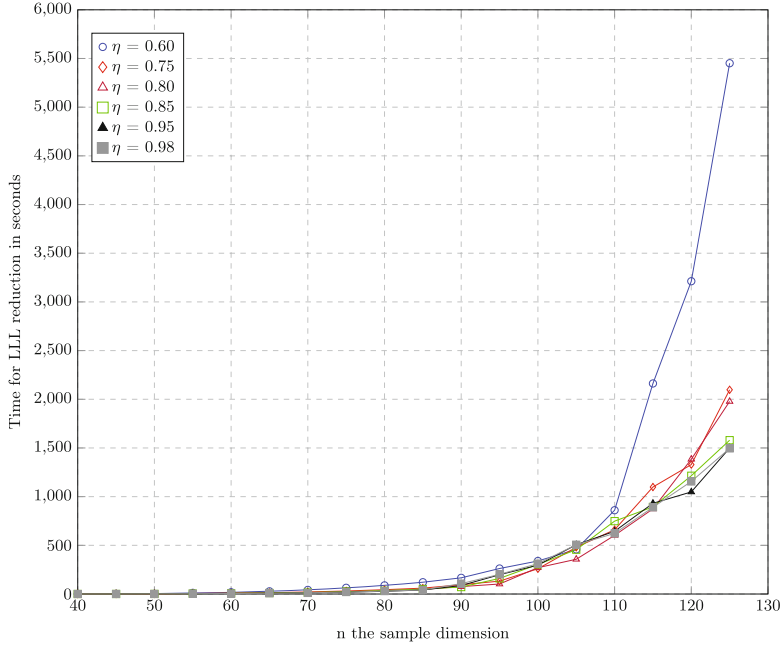


Fig. 1. Execution time in seconds in term of n for different η

Table 1. Observed reduction quality for different lattice parameters

γ	$\log_2 q = 47$	$\log_2 q = 95$
$n = 50$	1.00238	1.00129
$n = 200$	1.00300	1.00147

lattice point \mathbf{v}_0 , $\mathbf{w} - \mathbf{v}_0 = \begin{pmatrix} \sigma \mathbf{s} \\ \mathbf{e} \end{pmatrix}$, we can set the bound R_{2n} to the expected norm of $\begin{pmatrix} \sigma \mathbf{s} \\ \mathbf{e} \end{pmatrix}$. For a Gaussian error distribution we have with high probability $\|\mathbf{e}\|^2 \leq n \times (3\sigma^2)$, so $R_{2n}^2 = n\sigma^2 + n(3\sigma)^2 = 10n\sigma^2$.

In their paper, Gama et al. [GNR10] introduce three pruning functions: linear, step and piecewise linear, whose they study the resulting time complexity and success probability. They also mention another pruning function obtained by numerical optimization. In our case, we start with a linear pruning function defined by $R_k^2 = (k/2n)R_{2n}^2$.

One of the surprising finding of our experiments is that the enumeration terminates with the first candidate solutions, in nearly every cases. Recall that enumeration for BDD begins with getting a vector *quite close* to the target, and then, enumerates around, towards the one which minimizes the distance. So this first enumerated vector is equal to that outputed by Babai's Nearest Plane algorithm [Bab86]. It so happens that this solution falls below our pruning bound for most of our test cases, and leads to a successful key recovery. Both our implementation of BDD and `fp111`'s `ClosestVector` routine show this behavior.

A qualitative explanation for this unexpected costless enumeration is that the error norm (depending on σ) is very small compared to the Gram-Schmidt vectors, whose norms highly depend on q . Consequently, the simple rounding from Babai algorithm seems enough to get directly to the closest vector, component by component. This result stands also for other FHE schemes, in all which we have the same properties for σ and q .

5.3 Attacking Higher Dimensions

With these first conclusions we decided to go for a greater range of values for n and q . In terms of possible circuit depth evaluation, this range covers up to $L = 13$. We set the attack to an LLL reduction with $\delta = 0.99$ and $\eta \in \{0.51, 0.61, 0.71\}$ and an enumeration limited to Nearest Plane, and launched it for a few weeks.

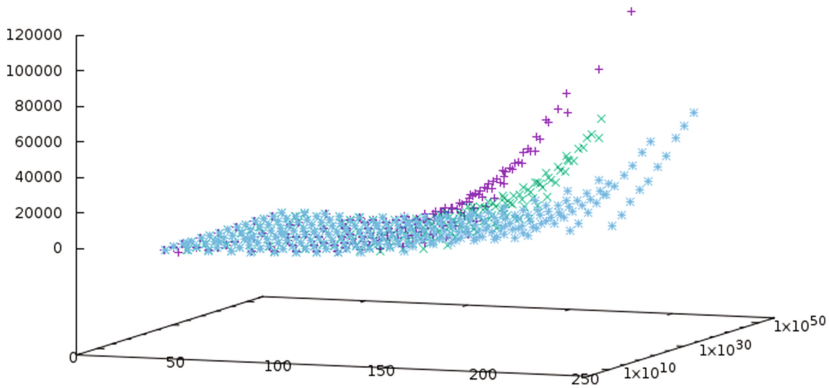


Fig. 2. Execution time in seconds in terms of n and q for different η

With n up to 250 we see in Fig. 2 that with $\eta = 0.71$ the attack takes fairly less time than 0.51, roughly 5 times less and still finishes successfully. This motivated us to keep only the version $\eta = 0.71$ and continue to higher dimensions.

In the end we were able to successfully break [FV12] keys with $n = 250$ and $\log q = 46.8$ in 10 h, or $n = 320$ and $\log q = 68.7$ in little less than 28 h, see Fig. 3. Our result compares favorably with the work of Laine and Lauter [LL15], who were able to recover a key in dimension 350 in 3.5 days, yet with a less generic attack working for q very large (2^{52}) and σ very small (3.2).

Such parameters are not considered secure, even prior to our work. One would take much greater n for instance. Yet they serve as good examples to understand the performance of our special-purpose attack. When composed with an LLL reduction and an enumeration with Nearest Plane, our proposal requires only a polynomial number of operations (in the parameters of the lattice) to complete. Moreover, the estimator from [APS15], which takes into account only generic attacks, including the latest [Alb17], predicts one month of computation to break such key.

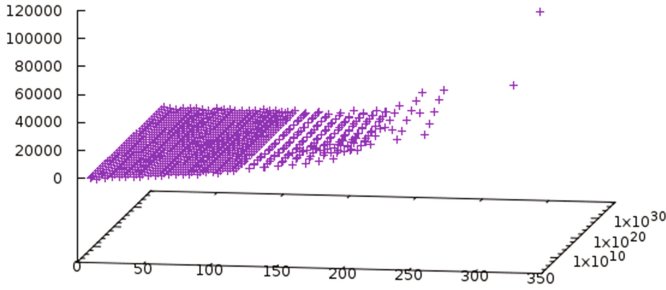


Fig. 3. Execution time in seconds in terms of n and q for $\eta = 0.71$

6 Conclusion

In this work, we aimed at assessing the practical security of FHE schemes based on Ring-LWE. To us, this is a topic that needs more focus, since the requirements of correctness in FHE lead to very special shape of Ring-LWE parameters. After reviewing state-of-the-art attacks, we presented a new special-purpose attack for the case at hands. Our experiments show that such attack has unexpected performance: with only a polynomial number of steps, it successfully breaks keys with parameters beyond toy sizes.

Our main results, on lattice reduction and enumeration in FHE cases, confirm our opinion that attacker may have unexpected advantage in special situations. Our result does not contradict the security reductions of Ring-LWE, but when picking practical parameters, it is really important to consider such results. The discussion about sizing parameters to guarantee a security level objective is far from being closed. We hope we raise interest for this kind of work which is of great importance to move forward with FHE implementation.

Acknowledgements. This work has been supported by the Chair of Naval Cyber Defense, funded by Ecole Navale, IMT-Atlantique, Thales and Naval Group.

References

- [AG11] Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 403–415. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22006-7_34
- [Alb17] Albrecht, M.R.: On dual lattice attacks against small-secret LWE and parameter choices in HELib and SEAL. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 103–129. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_4
- [AMGH10] Melchor, C.A., Gaborit, P., Herranz, J.: Additively homomorphic encryption with d -operand multiplications. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 138–154. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_8

- [APS15] Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of Learning with Errors. Cryptology ePrint Archive, Report 2015/046 (2015)
- [Bab86] Babai, L.: On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica* **6**(1), 1–13 (1986)
- [BG14] Bai, S., Galbraith, S.D.: Lattice decoding attacks on binary LWE. In: Susilo, W., Mu, Y. (eds.) ACISP 2014. LNCS, vol. 8544, pp. 322–337. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08344-5_21
- [BGV12] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, pp. 309–325. ACM (2012)
- [BKW03] Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM (JACM)* **50**(4), 506–519 (2003)
- [CGGI16] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: bootstrapping in less than 0.1 Seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_1
- [CIV16] Castryck, W., Iliashenko, I., Vercauteren, F.: Provably weak instances of ring-LWE revisited. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 147–167. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_6
- [CLS15] Chen, H., Lauter, K., Stange, K.E.: Attacks on Search RLWE. Cryptology ePrint Archive, Report 2015/971 (2015). <http://eprint.iacr.org/>
- [CN11] Chen, Y., Nguyen, P.Q.: BKZ 2.0: better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_1
- [Cry] CryptoExperts. FV-NFLlib. <https://github.com/CryptoExperts/FV-NFLlib>
- [DGBL+15] Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Manual for using homomorphic encryption for bioinformatics. Technical report MSR-TR-2015-87, November 2015
- [DM15] Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_24
- [DS15] Doröz, Y., Sunar, B.: Flattening NTRU for Evaluation Key Free Homomorphic Encryption (2015). <http://eprint.iacr.org/>
- [dt16] The FPLLL development team. fpLLL, a lattice reduction library (2016). <https://github.com/fplll/fplll>
- [EHL14] Eisenträger, K., Hallgren, S., Lauter, K.: Weak instances of PLWE. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 183–194. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13051-4_11
- [ELOS15] Elias, Y., Lauter, K.E., Ozman, E., Stange, K.E.: Provably weak instances of ring-LWE. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 63–92. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_4
- [FV12] Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive* 2012:144 (2012)

- [Gen09] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, vol. 9, pp. 169–178 (2009)
- [GHGN06] Gama, N., Howgrave-Graham, N., Nguyen, P.Q.: Symplectic lattice reduction and NTRU. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 233–253. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_15
- [GJS15] Guo, Q., Johansson, T., Stankovski, P.: Coded-BKW: solving LWE using lattice codes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 23–42. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_2
- [GN08a] Gama, N., Nguyen, P.Q.: Finding short lattice vectors within mordell’s inequality. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, pp. 207–216. ACM (2008)
- [GN08b] 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
- [GNR10] Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 257–278. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_13
- [GSW13] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5
- [Hal] Halevi, S.: Helib. <https://github.com/shaih/HElib>
- [HPS11] Hanrot, G., Pujol, X., Stehlé, D.: Algorithms for the shortest and closest lattice vector problems. In: Chee, Y.M., Guo, Z., Ling, S., Shao, F., Tang, Y., Wang, H., Xing, C. (eds.) IWCC 2011. LNCS, vol. 6639, pp. 159–190. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20901-7_10
- [Kan87] Kannan, R.: Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.* **12**(3), 415–440 (1987)
- [KGV15] Khedr, A., Gulak, G., Vaikuntanathan, V.: SHIELD: scalable homomorphic implementation of encrypted data-classifiers. *IEEE Trans. Comput.* **65**(9), 2848–2858 (2015)
- [Lep14] Lepoint, T.: A proof-of-concept implementation of the homomorphic evaluation of SIMON using FV and YASHE leveled homomorphic cryptosystems (2014). Accessed 18 Aug 2015
- [LL15] Laine, K., Lauter, K.: Key Recovery for LWE in Polynomial Time. *Cryptography ePrint Archive*, Report 2015/176 (2015)
- [LLL82] Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. *Math. Ann.* **261**(4), 515–534 (1982)
- [LN13] Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: an update. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 293–309. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36095-4_19
- [LN14] Lepoint, T., Naehrig, M.: A comparison of the homomorphic encryption schemes FV and YASHE. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT 2014. LNCS, vol. 8469, pp. 318–335. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06734-6_20

- [LP11] Lindner, R., Peikert, C.: Better key sizes (and Attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19074-2_21
- [LPR10] 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
- [MW15] Micciancio, D., Walter, M.: Practical, predictable lattice basis reduction. Cryptology ePrint Archive, Report 2015/1123 (2015). <http://eprint.iacr.org/>
- [NS05] Nguên, P.Q., Stehlé, D.: Floating-point LLL revisited. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_13
- [Pei15] Peikert, C.: A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939 (2015). <http://eprint.iacr.org/>
- [Pei16] Peikert, C.: How (Not) to Instantiate Ring-LWE. Cryptology ePrint Archive, Report 2016/351 (2016)
- [Reg05] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, pp. 84–93. ACM (2005)
- [SE94] Schnorr, C.-P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Math. Program.* **66**(1–3), 181–199 (1994)
- [Sho15] Shoup, V.: NTL - A Library for doing Number Theory (2015). Accessed 18 Aug 2015
- [SL96] Storjohann, A., Labahn, G.: Asymptotically fast computation of hermite normal forms of integer matrices. In: Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, pp 259–266. ACM (1996)
- [vDGHV10] van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2
- [vdPS13] van de Pol, J., Smart, N.P.: Estimating key sizes for high dimensional lattice-based systems. In: Stam, M. (ed.) IMACC 2013. LNCS, vol. 8308, pp. 290–303. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45239-0_17

Architecture Level Optimizations for Kummer Based HECC on FPGAs

Gabriel Gallin¹, Turku Ozlum Celik², and Arnaud Tisserand³(✉) 

¹ CNRS, IRISA UMR 6074, INRIA Centre Rennes - Bretagne Atlantique and University Rennes 1, Lannion, France
`gabriel.gallin@irisa.fr`

² IRMAR, University Rennes 1, Rennes, France
`turku-ozlum.celik@univ-rennes1.fr`

³ CNRS, Lab-STICC UMR 6285 and University South Brittany, Lorient, France
`arnaud.tisserand@univ-ubs.fr`

Abstract. On the basis of a software implementation of Kummer based HECC over \mathbb{F}_p presented in 2016, we propose new hardware architectures. Our main objectives are: definition of architecture parameters (type, size and number of units for arithmetic operations, memory and internal communications); architecture style optimization to exploit internal parallelism. Several architectures have been designed and implemented on FPGAs for scalar multiplication acceleration in embedded systems. Our results show significant area reduction for similar computation time than best state of the art hardware implementations of curve based solutions.

Keywords: Hyper-elliptic curve cryptography
Hardware implementation · Architecture exploration
Embedded systems

1 Introduction

Reducing the cost of asymmetric cryptography is a challenge for hardware implementation of embedded systems where silicon area is limited. Hyper-elliptic curve cryptography (HECC [6]) is considered to be an interesting solution compared to elliptic curve cryptography (ECC [12]). HECC requires smaller finite fields than ECC at similar security level. For instance, size of field elements is divided by two in genus-2 HECC solutions. But the number of field-level operations is larger in HECC per key/scalar bit. Then comparisons depend a lot on curve parameters, algorithm optimizations and implementation efforts.

HECC solutions based on Kummer surfaces (see [10] for details) demonstrate promising improvements for embedded software implementations. In 2016, Renes *et al.* presented in [24] a new Kummer-based HECC (KHECC) solution and its implementation on microcontrollers with 30 to 70% clock cycles count reduction compared to the best similar curve based solutions at equivalent security level.

To the best of our knowledge, there is no hardware implementation of this recent KHECC solution. Below, we present hardware architectures for KHECC

adapted from [24] for scalar multiplication and their FPGA implementations. We study and evaluate the impact of various architecture parameters on the cost and performances: type, size and number of units (arithmetic, memory, internal communications); architecture topology; and exploitation of internal parallelism. We target embedded applications where the FPGA bitstream cannot be changed easily (trusting the configuration system at application level is complex) or prototyping for ASIC applications. Then to provide flexible circuits at software level, our solutions are designed for \mathbb{F}_p with generic primes (where [24] only deals with $p = 2^{127} - 1$). Several architectures have been designed and implemented on different FPGAs for various parameters, and compared in terms of area and computation time.

Our paper outline is as follows. Section 2 recalls background on KHECC and introduces notations. Section 3 presents major elements of the work [24] used as a starting point and discusses required adaptations for hardware implementation. Section 4 quickly presents our units selected for the architectures and our tools used for design space exploration. Section 5 describes the proposed KHECC architectures and their implementation results on different FPGAs. Section 6 reports comparisons. Finally, Sect. 7 concludes the paper.

2 State-of-the-Art

HECC was introduced by Koblitz in [14] as a larger set of curves compared to ECC with a generalization of the class of groups obtained from the jacobians of hyper-elliptic curves. Subsequently, many HECC improvements have been proposed. See book [6] for a complete presentation on HECC and book [12] for ECC. Broadly speaking, field elements in HECC are smaller than in ECC for a similar security level (*e.g.* 128-bit HECC on genus-2 curves is equivalent to 256-bit ECC). This reduction should directly benefit to HECC since the width of field elements has a major impact on circuit area. But HECC requires more field operations to achieve operations at curve level such as point addition (ADD) and point doubling (DBL) for each scalar/key bit.

Many efforts have been made to reduce the cost of curve level operations in HECC. For genus-2, one can refer to Lange [18], Gaudry [10], Bos *et al.* [5] and Renes *et al.* [24] for instance. Table 1 reports a few costs for HECC and ECC solutions. There were many works on \mathbb{F}_{2^n} solutions at low security levels (fields with 80–90 bits) in the past but very few on \mathbb{F}_p at 128-bit security level until recently in software (our goal is hardware implementation).

KHECC solutions from [24] are based on a Kummer surface \mathcal{K}_C of an hyper-elliptic curve \mathcal{C} defined over \mathbb{F}_p . Curve \mathcal{C} is defined using constant parameters among which the “squared theta constants” (a, b, c, d) used during scalar multiplications. Points are represented by tuples of four n -bit coordinates in \mathbb{F}_p where $\pm P = (x_P : y_P : z_P : t_P)$ is the projection of P from \mathcal{C} on \mathcal{K}_C .

In (H)ECC primitives such as signature or key exchange, the main operation is the *scalar multiplication* $[k]P_b$ of a base point P_b by a m -bit scalar or key k . In embedded systems, scalar multiplication must be protected against *side channel*

Table 1. Cost per key bit of curve level operations in various (H)ECC solutions (M and S denote multiplication and square in the finite field).

Solution & source	Field width [bit]	ADD	DBL
\mathbb{F}_p ECC [4]	ℓ_{ECC}	12M + 2S	7M + 3S
\mathbb{F}_{2^n} HECC [18]	$\ell_{\text{HECC}} \approx 0.5\ell_{\text{ECC}}$	40M + 4S	38M + 6S
\mathbb{F}_p KHECC [24]	$\ell_{\text{HECC}} \approx 0.5\ell_{\text{ECC}}$	19M + 12S	

attacks (SCAs [20]). A popular protection against SCAs is the adaptation of *Montgomery ladder* (ML) algorithm [22]. ML is *constant time* (i.e. computation time of iterations does not depend on the key bit values) and *uniform* (i.e. the exact same schedule of the exact same field operations is executed at each iteration whatever the key bit values).

For KHECC hardware implementations (but also for more general HECC solutions), designers have to face several questions. How one should exploit the internal parallelism available at field level? Are few large and fast units more efficient than several parallel small and slow units? How to select parameters in a parallel architecture? Our work was related to those questions.

3 Hardware Adaptation of Renes *et al.* Solution

We based our work on the KHECC solution presented by Renes *et al.* at CHES 2016 [24] for software implementations of Diffie-Hellman key exchange and signature at 128-bit security level. Their solution optimizes the use of Kummer surface of hyper-elliptic curve described by Gaudry [10].

3.1 Analysis of Renes *et al.* Solution

In [24], the prime for \mathbb{F}_p is $p = 2^{127} - 1$ due to fast modular reduction algorithms for Mersenne primes. The scalar size is $m = 256$ bits. ML algorithm starts with most significant key bits first. Each iteration computes a couple of points $(\pm V_1, \pm V_2)$ of \mathcal{K}_C from the result of the previous iteration using curve level operations CSWAP and xDBLADD. Using initial values $\pm V_1 = (a : b : c : d)$ and $\pm V_2 = (x_{P_b} : y_{P_b} : z_{P_b} : t_{P_b})$, the scalar multiplication computes $(\pm[k]P_b, \pm[k+1]P_b)$.

The core operation in ML iterations is the modified pseudo-addition xDBLADD *combined differential double-and-add* (see [24]). Given points $\pm V_1, \pm V_2$ on \mathcal{K}_C , and base point $\pm P_b$, it computes $(\pm[2]V_1, \pm(V_1 + V_2)) = \text{xDBLADD}(\pm V_1, \pm V_2, \pm P_b)$. Based on the set of \mathbb{F}_p operations described in Fig. 1, xDBLADD has a *constant time* and *uniform* behavior.

The CSWAP operation consists in swapping the 2 input points (IN) of xDBLADD and the 2 resulting points (OUT) depending on the current key bit value (see Algorithm 7 in [24]). It does not involve any computation but it impacts SCA aspects.

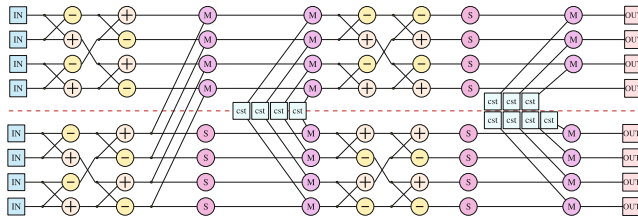


Fig. 1. \mathbb{F}_p operations in xDBLADD from [24] (IN/OUT are $\pm V_1, \pm V_2$ coordinates).

Renes *et al.* performed a smart selection of optimized curve parameters (from [10]) to determine constants with reduced size: 16 bits instead of 127. Then they use a dedicated optimized function for modular multiplication by this type of constants in the software implementation.

Their implementations target low-cost microcontrollers: 8-bit AVR AT Mega and 32-bit ARM Cortex M0. They report significant improvements over the best known solutions. On Cortex M0, the clock cycles count is reduced by 27% for key exchange and by 75% for scalar multiplication in signature. On AT Mega, the corresponding reductions are respectively 32% and 71%.

3.2 Objectives and Constraints for Our Hardware Accelerators

Unlike Renes *et al.* [24], in the present paper we only propose hardware acceleration for scalar multiplication since this is the main operation in terms of performance, energy consumption and security against SCAs (when the scalar is the private key). As is frequently the case in a complete embedded system, we assume that our hardware accelerator is coupled to a software implementation for high level primitives (which are out of scope of this paper).

In order to design flexible hardware accelerators and to report results in a general case, we target KHECC on generic prime fields. In [24] the selected prime $p = 2^{127} - 1$ leads to very cheap modular reduction but it is very specific (there is no Mersenne prime for slightly different security levels). Currently, we only deal with generic primes but we plan to derive versions for specific ones (*e.g.* pseudo-Mersenne) in the future. Field characteristic impacts the choice of curve parameters. We propose to use material presented in the work from Gaudry [10] to derive curves parameters and implementation constants.

One of our goal is to study hardware accelerators for scalar multiplication at architecture level. KHECC offers some internal parallelism as illustrated in Fig. 1. Groups of 4 to 8 \mathbb{F}_p operations can be easily performed at the same time with uniform and constant time schedules. In most of ECC solutions, fewer operations can be performed in parallel. We will evaluate the impact of this parallelism on the design of efficient accelerators with various trade-offs in terms of area and computation time (see questions at end of Sect. 2).

This paper is not dedicated to protection against physical attacks. But we target hardware accelerators where the execution of ML type of algorithms is

actually constant time and uniform at low level. We will describe how we designed some units to achieve this objective (not yet evaluated using real attacks).

In order to provide hardware accelerators that can be easily adapted to other application constraints and algorithms, we use a modular type of architectures based on independent units (for arithmetic operations and internal storage); a microcoded control; and an internal communication system based on multiplexors (between some units). This type of modular architectures allows us to easily explore many solutions in design space: type, size and number of units; size of internal communications; scheduling impact on performances and security. This choice also comes from the type of dedicated resources available in modern FPGAs: small embedded multipliers (DSP slices/blocks), small embedded memory (BRAM: block RAM), dedicated multiplexors in routing resources. Architecture modularity also helps the debug and validation process of the proposed solutions using a hierarchical method. First, we extensively and individually simulate in HDL and evaluate on FPGAs all our units. Second, we extensively simulate in HDL and evaluate on FPGAs the complete accelerator solutions.

4 Accelerator Units and Exploration Tools

Architecture modularity allows to explore a wide parameter space but it requires huge efforts in terms of implementation and debug. First, we decided to design, implement, validate, and optimize a small set of units for arithmetic operations, memory and internal communications. They are presented in Sect. 4.1. Second, based on this set of units, we designed specific tools for architecture level exploration and evaluation. These tools are quickly described in Sect. 4.2.

4.1 Accelerator Units and Resources

There are several types of resources in our hardware accelerators:

- arithmetic units for field level operations (\mathbb{F}_p addition, subtraction and multiplication with generic prime p in this work);
- memory unit(s) for storing intermediate values (\mathbb{F}_p elements for points coordinates), curve parameters and constants;
- a CSWAP unit in charge of scalar management and on-line schedule of operands depending on scalar/key bit values;
- an internal communication system for data transfers between the units;
- a control based on a microcode running the architecture.

Multiplier (Mult): Prime p is generic and can be programmed in our architecture. Then, there is no low-cost modular reduction. We use the *Montgomery modular multiplication* (MMM) proposed in [21]. Operands and product are represented in *Montgomery domain* (MD) $\{(x \times R) \bmod p, \forall x \in \mathbb{F}_p\}$ with $\gcd(R, p) = 1$ and $R > p$. We use MMM optimization methods from Orup [23]

and Koc *et al.* [15] where internal high-radix computations reduce the amount of data dependencies and partial product/reduction steps are interleaved to speed up the multiplication with a slightly larger internal datapath.

In our target Xilinx FPGAs (see Sect. 5), there are embedded multipliers for 18×18 bits signed integers called DSP blocks/slices (in 2's complement representation). But for \mathbb{F}_p computations, one can only use the 17 LSBs (all but the “sign” bit). Then our datapath width must be a multiple of 17. We evaluated that 34-bit word size for multiplication is interesting in our KHECC context (smaller size leads to slow multipliers, larger size requires too many DSP slices). For 34-bit words, operands and products are between 0 and $2p$ represented using 136 bits in MD (136 is the closest multiple of 17 and 34 larger than 127). Then we can select $R > 4p$ for speed purpose (as many works in state of the art).

For improving the efficiency of our accelerators, we used the *hyper-threaded* multiplier version we proposed in [9] specially for KHECC. Hyper-threading hides the latency in DSP slices at high frequency (when all internal registers are activated). It computes 3 independent MMMs in parallel using 11 DSP slices (17×17), 2 BRAMs and a few slices in 79 clock cycles at 360 MHz on Virtex 5.

Due to MD, all field elements, parameters and constants are 128-bit values (contrary to [24] where shorter constants can be used). Using shorter constants in hardware requires a specific unit and then a more complex control and smaller overall frequency. Also for flexibility and frequency reasons, we only use generic \mathbb{F}_p multiplier `Mult`. For the same reasons, we avoid dedicated square units (for generic p , MMM variants for square operation only lead to small improvements).

Similarly to state of the art solutions, final reduction from MD ($0 \leq x < 2p$) to “standard” \mathbb{F}_p ($0 \leq x < p$) is performed after the scalar multiplication (there is no performance or security issue for this conversion).

Adder (AddSub): It performs both modular addition/subtraction ($(x \pm y) \bmod 2p$) in MD by setting a mode signal at operation start-up. Subtraction is implemented by adding operand x with the 2's complement of y operand (\bar{y}). The reduced sum in MD (resp. difference), is obtained from the parallel computations $r = x + y$ (resp. $r = x + \bar{y}$) and $r_p = r + \overline{2p}$ (resp. $r_p = r + 2p$). The output is r if $0 \leq r < 2p$, else r_p . The range of r is determined from the value of the output carry bit of $r + \overline{2p}$ for addition and of $x + \bar{y}$ for subtraction. We evaluated the impact of `AddSub` units for several word sizes: 34, 68 and 136 bits. The two large ones significantly reduce the overall frequency of the accelerator (due to longer carry propagations). It seems that our target FPGAs are optimized to handle word sizes around 32 bits but not for larger widths without costly pipeline schemes. To enforce short combinatorial paths and simplify the control, we set `AddSub` internal datapath width to $w_{arith} = 34$ bits as in `Mult`.

Memory: The accelerator uses internal memory(ies) for storing intermediate values (\mathbb{F}_p elements of points coordinates), curve parameters and some constants (*e.g.* initial values ($a : b : c : d$)). To fit the internal width of arithmetic units ($w_{arith} = 34$ bits) and BRAM width in Xilinx FPGAs (configurable into

Table 2. Memory and internal communication width configurations.

Config	w [bit]	s [word]	Cycle(s)/mem. op.	BRAM(s)
w34	34	4	4	1
w68	68	2	2	2
w136	136	1	1	4

$\{1, 2, 4, 9, 18, 36\}$ bits), we selected a memory configuration where words are multiple of 34 bits to simplify the control and avoid interfaces. Due to the large number of memory operations (read/write), we will show in Sect. 5 that a wider memory reduces the number of clock cycles per memory operation of full 136-bit values. We tested 3 configurations for the memory width (and internal communications see below) described in Table 2. The main parameter w (in bits) is the internal width of memory words (and communications). Related parameter s is the number of words required for storing a complete 136-bit value. Our BRAMs are configured into 512 lines of 36 bits words (2 unused bits per word). Less than 512 words are required for KHECC even in **w34** configuration. Table 2 also reports the clock cycles count of each memory operation and the memory area (in BRAMs). For security reason, this internal memory is restricted to the accelerator and cannot be accessed from outside (inputs and outputs are handled by a specific very small unit which is mute during scalar multiplications).

Internal Communications: The units are interconnected through a specific internal communication system based on multiplexors (buses are not very efficient in target FPGAs and lead to high capacitances switching which can be a bad point for SCA protection). The communication system will be described in Sect. 5. In order to explore cost and performance trade-offs, we used configurations from Table 2 for the width in the internal communication system. Different widths for memory and communications requires a very costly control. Then w is shared for memory and communications. But for arithmetic units in this paper, we evaluated that $w_{\text{arith}} = 34$ is the best choice for our KHECC accelerators. For **w68** and **w136** configurations, small serial-parallel interfaces are added in the arithmetic units to handle the width difference with communications.

CSWAP Unit: In algorithm 7 from [24] (called `crypto_scalarmult`), the CSWAP operation manages the scalar/key bits by swapping, or not, points $\pm V_1$ and $\pm V_2$ at the beginning and end of each ML iteration. We designed a dedicated unit for this purpose. It reads $2 \mathbb{F}_p$ elements as inputs (corresponding to one coordinate of $\pm V_1$ and $\pm V_2$) and swaps them, or not, depending on the actual key bit value for the current iteration. At the last iteration, CSWAP triggers a “end of scalar multiplication” signal. In our CSWAP unit, there is no variable addresses or key management in the accelerator control for security reasons (instructions decoding does not depend on secret bits). For SCA protection, our CSWAP unit has

been designed to ensure that there is always electrical activity in the pipelined unit, communication system and memory between successive clock cycles even if there is no swapping (read/write operations for one coordinate are interleaved with those of the other coordinates). Our **CSWAP** unit (pipelined with internal communications and memory) has a constant time and uniform behavior.

In the future, we will investigate the use of advanced scalar recoding schemes on the performances and security against SCAs.

Accelerator Control: We defined a tiny ISA (instruction set architecture). Instructions, detailed in Table 3, are read from the code memory and decoded to provide control signals to/from units, communication system and memory. The user program is stored into a small program memory (one BRAM). This type of control provides flexibility, avoids long synthesis and place & route processes (during modifications of user programs), and leads to fairly high frequencies on the target FPGAs when using pipelined BRAMs and decoding.

Table 3. Instructions set for our accelerators.

Instruc.	Description
read	Transfer operands from memory to target unit and start computation
write	Transfer result from target unit to memory
wait	Wait for immediate clock cycles
nop	No operation (1 clock cycle)
jump	Change program counter (PC) to immediatecode address
end	Trigger the end of the scalar multiplication

Instructions are 36-bit wide and our KHECC programs fit into one single BRAM (<512 instructions). Instructions contain: 4-bit opcode, 3-bit unit index, 2-bit operation mode, two 9-bit memory addresses and 9-bit immediate value.

We implemented hardware loops, using small finite state machines (FSMs), to handle s cycles during communication and memory operations and duration of **wait** instruction. Instruction decoding does not handle or depend on the scalar bit values for SCA protection (only the **CSWAP** unit handles secret bits).

Control resources include a few w -bit registers dedicated to external communications and initialization of memory parameters. They are very small and not involved during scalar multiplications, then we do not detail them here.

In the future, we plan to explore other types of control (*e.g.* distributed or FSM based solutions without microcode) for ASIC implementations.

4.2 Tools for Exploration and Evaluation at Architecture Level

Several parameters must be specified at design time for each architecture:

- type and number of units (AddSub, Mult, CSWAP, memory);
- width w for internal memory and communications;
- topology of the architecture.

All units have been fully described in synthesizable VHDL for FPGA implementation (with optimizations for DSP slices and BRAMs). They can also be tested and evaluated using *cycle accurate and bit accurate* (CABA) simulations. Then the time model at every clock cycle and the hardware cost of each unit are perfectly known (from implementation results).

Fully designing all possible architectures in VHDL is too time consuming. We decided to define and use a hierarchical and heterogeneous method to efficiently explore and validate numerous architectures.

Each architecture is described and simulated using a high-level model based on a CCABA (critical CABA) specification¹. The *critical cycles* at architecture level are clock cycles where there are transitions in the control signals to/from the units and their inputs/outputs. For functional units, this corresponds to operands inputs, operation mode selection, start of computation, end of computation, and results outputs. The purely internal control signals inside the units are not modeled in CCABA (since their behavior is perfectly determined in the VHDL description and does not impact other parts of the accelerator).

A CCABA simulation tool has been developed in Python. Each unit is modeled in Python to specify: (a) its mathematical behavior and (b) its behavior at critical cycles based on the corresponding VHDL model (*e.g.* computation duration after start signal). For each unit, we need its complete VHDL and Python CCABA models (both manually written). Our tool allows fast simulations of complete scalar multiplications due to the hierarchical approach.

We also started the development of a tool to automatically schedule arithmetic and memory operations as well as internal communications. Currently, it uses a basic greedy algorithm to first feed the multipliers (due to their longer latency). We plan to improve it in the future. The schedule gives the total clock cycles count of a complete scalar multiplication.

We are able to quickly estimate the area and computation time of various architectures. The estimated area sums up the VHDL results for all units instantiated in the accelerator. The computation time is estimated by the total number of clock cycles multiplied by the slowest unit period. We approximate the impact of the control system (not yet designed in VHDL at this stage of the exploration) based on our experience.

During the exploration, we perform this type of estimation for each accelerator configuration to be evaluated. Then we select the most interesting solutions

¹ Our CCABA model is inspired by Transaction Level Modeling (TLM) with full cycle accuracy for all control signals at the architecture level but not inside the units (when there is no input/output impact).

for full implementation in VHDL, final validation, and accurate comparisons (the corresponding results are presented in Sect. 5).

Once the accelerator has been fully implemented in VHDL, it is intensively tested using both VHDL simulations and executions on FPGA cards against reference values computed by SAGE mathematical software.

5 Proposed Architectures

We explored various configurations for parameters and architectures. We selected 4 architectures summarized in Table 4 and fully implemented them in VHDL on several FPGAs. The corresponding results are reported in sub-sections below. We began with a small and basic architecture A1 embedding the minimum number of units. Then we explored optimizations and more parallel architectures. Architecture A2 uses an optimization of CSWAP unit (V2). Architecture A3 embeds 2 operators for each arithmetic operation (\pm and \times) to reach a higher parallelism (notice that a single `Mult` already handles 3 sets of operands in parallel using hyper-threading, see [9]). Architecture A4 is a cluster of parallel units for both arithmetic operations and data memory operations, and V3 of CSWAP unit.

Table 4. Main characteristics of the 4 implemented and evaluated architectures.

Resources	Architectures			
	A1 (Sect. 5.1)	A2 (Sect. 5.2)	A3 (Sect. 5.3)	A4 (Sect. 5.4)
<code>AddSub</code>	1	1	2	2
<code>Mult</code>	1	1	2	2
<code>CSWAP</code>	1 V1	1 V2	1 V2	1 V3
Data memory	1	1	1	2
Communication system	1	1	1	2 with bridge
Program memory	1	1	1	1
Control	1	1	1	1

The Xilinx FPGAs listed in Table 5 were our implementation targets. ISE 14.7 tools were used for synthesis and place & route, as well as SmartXplorer. V4/V5 FPGAs were used for comparison with state of the art. FPGA S6 was used for low-cost solutions and imminent SCA evaluation on SAKURA card [16]. In order to fairly compare area results, it should be remembered that slice and look-up table (LUT) definition strongly depends on the FPGA family, see examples in Table 5. Flip-flop (FF) means a 1-bit register. One LUT6 is equivalent to 4 LUT4. Then slices in V4 or in V5/S6 should not be compared directly.

For architectures A1–4, we report below implementation results for `w34`, `w68`, `w136` configurations on V4, V5, S6 FPGAs using 100 SmartXplorer runs.

Table 5. Target FPGAs with some characteristics (f_{\max} is the maximum frequency).

Short name	Model	Techno. [nm]	Slice content		f_{\max} DSP [MHz]	BRAM capacity
			LUT	Flip-flop		
V4	Virtex 4 VLX100	90	2 LUT4	2	500	18 Kb
V5	Virtex 5 LX110T	65	4 LUT6	4	550	36 Kb
S6	Spartan 6 SLX75	45	4 LUT6	4	390	18 Kb

5.1 Architecture A1: Base Solution

Architecture A1, depicted in Fig. 2, corresponds to a basic Harvard processor dedicated to the scalar multiplication derived from [24] with our modifications detailed in Sect. 3.2. This is the smallest accelerator with only one instance of each type of unit (AddSub, Mult, and CSWAP-V1 described in Sect. 4.1).

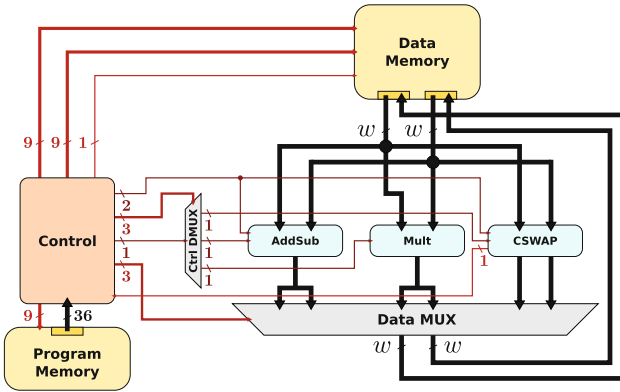


Fig. 2. Architecture A1 with its main units (arithmetic and memory), internal communication system, and control (warning: areas of boxes are not on a real scale).

Architecture A1 was fully implemented in VHDL for the 3 widths $w \in \{34, 68, 136\}$ on 3 FPGAs $\{V4, V5, S6\}$. The corresponding results are reported in Table 6. Intensive VHDL simulations were used for validation.

Table 6 reveals a few trends:

- Width w has a small impact (at most 5% reduction) on the clock cycles count since most of the time is spent into the single AddSub and Mult.
- Width w strongly impacts the area in LUTs (+50–70% for $w68$ compared to $w34$, and +80–110% for $w136$ compared to $w34$ depending on the FPGA). Clearly, enlarging the datapath requires more logic cells.
- The link between w and flip-flops numbers seems tricky. This is partly due to the cost of serial-parallel interfaces between arithmetic operators ($w_{\text{arith}} = 34$) and memory/communications ($w68$ the most complex case, and $w136$).

Table 6. FPGA implementation results for architecture A1 (all BRAMs are 18 Kb ones, only 17×17 multipliers were used in DSP slices for all FPGAs).

FPGA	w [bit]	LUT	FF	Logic slices	DSP blocks	RAM blocks	Freq. [MHz]	Clock cycles	Time [ms]
V4	34	1010	1833	1361	11	4	322	194,614	0.60
	68	1750	3050	2251	11	5	305	186,911	0.61
	136	2281	3028	1985	11	7	266	184,337	0.69
V5	34	757	1816	603	11	4	360	194,614	0.54
	68	1264	3033	908	11	5	360	186,911	0.52
	136	1582	3008	940	11	7	360	184,337	0.51
S6	34	1064	1770	408	11	4	278	194,614	0.70
	68	1555	2970	705	11	5	252	186,911	0.74
	136	1910	2994	747	11	7	221	184,337	0.83

- The increase in BRAMs comes from the wider memory for configurations **w68** and **w136** (see Table 2).
- Frequency decreases when w increases due to longer combinatorial delays and larger fanout. Depending on the FPGA, the reduction varies about 5–20%. On V5, 360 MHz is the frequency for both the slowest unit and the complete accelerator (the control does not impact the overall frequency for a small accelerator which uses a small part of the complete FPGA).

As a conclusion for our smallest architecture, using large w is not interesting. The reduction of the clock cycles count is canceled by the frequency drop for **w68** or **w136**. Hence, the best solution is always **w34** for A1 on all tested FPGAs.

5.2 Architecture A2: CSWAP Optimization

Architecture A2 is similar to A1 where we modified the CSWAP unit, version V2 (the architecture schematic is the same as Fig. 2). The ML algorithm proposed in [24] uses one CSWAP operation at the end of each iteration and another CSWAP at the beginning of the next iteration with the same key bit operands. As there is no computation between these 2 consecutive CSWAP operations, we propose to merge them (this halves the calls to the CSWAP unit).

Our *modified CSWAP-V2* uses 2 consecutive key bits: k_i and k_{i-1} (scalar k is used starting MSB first). There is no swapping when $k_i = k_{i-1}$, and swapping when $k_i \neq k_{i-1}$ (we just need one xor gate). The very first CSWAP-V2 call is computed using bits 0 (current bit) and k_{m-1} (“next” bit and MSB of k). The proposed modification does not change security aspects against SCAs. The accelerator is still constant-time and uniform. As for CSWAP in A1, we designed CSWAP-V2 with a uniform activity pipeline (see CSWAP description in Sect. 4.1).

Complete implementation results for A2 are reported in Table 7. A2 shows a similar behavior than A1 with respect to w variations. A few elements can be noticed for A2 as summarized below:

Table 7. FPGA implementation results for architecture A2 (all BRAMs are 18 Kb ones, only 17×17 multipliers were used in DSP slices for all FPGAs).

FPGA	w [bit]	LUT	FF	Logic slices	DSP blocks	RAM blocks	Freq. [MHz]	Clock cycles	Time [ms]
V4	34	872	1624	1121	11	4	330	184,374	0.56
	68	1556	2637	1978	11	5	290	183,071	0.63
	136	2161	3027	2100	11	7	327	183,057	0.56
V5	34	722	1605	541	11	4	360	184,374	0.51
	68	1196	2620	840	11	5	360	183,071	0.51
	136	1419	3009	944	11	7	360	183,057	0.51
S6	34	940	1559	381	11	4	293	184,374	0.63
	68	1503	2565	553	11	5	262	183,071	0.70
	136	1890	2981	667	11	7	283	183,057	0.65

- Clock cycles count in A2 is slightly smaller than A1 due to reduced number of CSWAP operations.
- Frequency is slightly higher for large w compared to A1. Frequency variations are smaller in A2 than A1.
- Computation time in A2 is slightly smaller than A1: $-5-10\%$ depending on the FPGA. The best solution is obtained for small w (the 0.8% speed-up for $w68/w136$ in V5 is not relevant due to the large area increase).
- Area (LUTs, FFs and slices) in A2 is slightly smaller than A1 due to a simplified management of CSWAP operations: $-5-13\%$ depending on the FPGA.
- DSP slices and BRAMs are identical in A2 and A1 (not related to CSWAP-V2).

As a conclusion for architecture A2, the best configuration is always $w34$ for all tested FPGAs. This optimization is interesting since A2 is slightly more efficient than A1 in terms of both speed and area (about 10%).

5.3 Architecture A3: Large Architecture

Architecture A3, depicted in Fig. 3, embeds more arithmetic units: 2 `AddSub` and 2 `Mult` units. It can perform up to 6 \mathbb{F}_p multiplications in parallel (only 3 for A1/A2) using our hyper-threaded `Mult` unit (see Sect. 4.1 and [9]).

Complete implementation results for A3 are reported in Table 8. A3 behaves quite differently from A1/A2.

- Adding 1 `AddSub` and 1 `Mult` increases LUTs by 60–90% depending on the FPGA and w . The second `Mult` adds 11 DSP slices and 2 BRAMs. This confirms that \mathbb{F}_p units constitute the largest resources in the accelerator.
- Frequency is slightly smaller in A3 than A2 due to larger fanout and more complex control. The frequency drop for increasing w values is very small for V4/V5 (less than 4%), and about 15% for S6.
- Unlike A1/A2, w has a large impact on the clock cycles count in A3: 34% reduction for $w68$ compared to $w34$ and 36% for $w136$. More arithmetic operations in parallel put pressure on the memory and communication system. A larger w allows to actually exploit more parallelism.

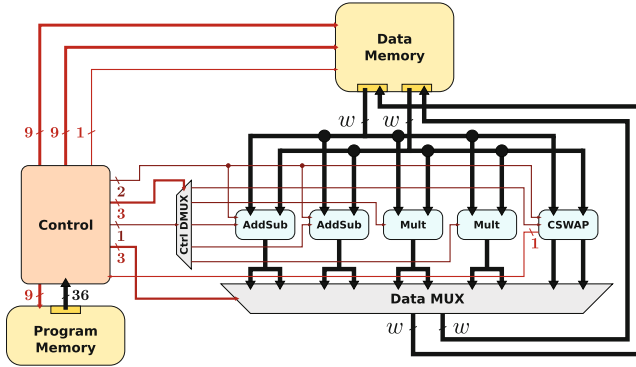


Fig. 3. Architecture A3 with its main units (arithmetic and memory), internal communication system, and control (warning: areas of boxes are not on a real scale).

Table 8. FPGA implementation results for architecture A3 (all BRAMs are 18 Kb ones, only 17×17 multipliers were used in DSP slices for all FPGAs).

FPGA	w [bit]	LUT	FF	Logic slices	DSP blocks	RAM blocks	Freq. [MHz]	Clock cycles	Time [ms]
V4	34	1462	2611	1783	22	6	294	188,218	0.64
	68	2802	4367	3468	22	7	282	124,191	0.44
	136	3768	5017	3660	22	9	285	119,057	0.42
V5	34	1262	2607	921	22	6	358	188,218	0.53
	68	2290	4403	1409	22	7	345	124,191	0.36
	136	2737	4978	1594	22	9	348	119,057	0.34
S6	34	1527	2503	668	22	6	265	188,218	0.71
	68	2421	4267	1020	22	7	225	124,191	0.55
	136	3007	4877	1131	22	9	225	119,057	0.53

- Computation time benefits from the reduction of clock cycles count for large values of w : 25 to 35% reduction for $w136$ depending on the FPGA.
- A3 is faster than A2: from 16 to 35% depending on the FPGA. But this speed-up comes at the expense of a larger area.

As a conclusion for architecture A3, there is no best solution but various compromises in terms of area and speed. When area is limited, $w34$ is interesting but computation time is 25–33% larger. When speed is the main objective, $w68$ and $w136$ lead to the fastest solutions but the area overhead is important.

5.4 Architecture A4: Clustered Architecture

A closer look at Fig. 1 shows that $x\text{DBLADD}$ can be decomposed into 2 clusters of \mathbb{F}_p operations with few dependencies using the red dashed horizontal line (*i.e.* one cluster above, one cluster below). Only 4 values have to be transferred from bottom cluster to top one at each ML iteration.

Architecture A4, depicted in Fig. 4, was designed to exploit this decomposition using a clustered accelerator. It also embeds a new optimization of the

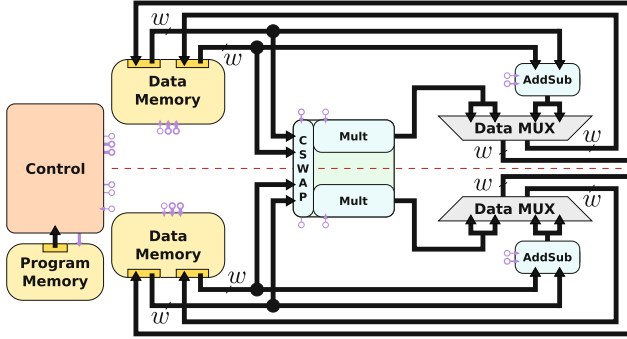


Fig. 4. Architecture A4 with its 2 clusters (units, data memory and local communication system), common control and new modification of CSWAP (warning: areas of boxes are not on a real scale).

CSWAP unit described below. To lighten Fig. 4, control signals are not completely drawn but represented by small circles. Constants values are duplicated in each cluster memory when necessary (at no cost in BRAMs).

We added a new modification of the CSWAP behavior (V3). CSWAP operation is replaced by 2 new swapping operations CS_0 and CS_1 (with $4 \mathbb{F}_p$ operands):

- $CS_0(A, B, C, D)$ returns (A, B, C, B) if $k_i = 0$, else it returns (C, D, A, D)
- $CS_1(A, B, C, D)$ returns (A, B, C, D) if $k_i = 0$, else it returns (C, D, A, B) .

The modification of the CSWAP behavior (V3) was associated to a new schedule for x DBLADD presented in Fig. 5. This figure is simplified, each black line now represents the communication of 4 operands from Fig. 1. H box represents a set of $8 \mathbb{F}_p$ additions/subtractions, and M box a set of $4 \mathbb{F}_p$ multiplications (only 3 in the top right upper box of Fig. 5). CS_0 and CS_1 are respectively in charge of the first and last CSWAP-V3 of the original ML iteration. Square operations of the original x DBLADD have been replaced by multiplications in the new solution (*i.e.* $A^2 \times B$ is now $A \times B \times A$) since we do not implement dedicated square units. This does not change the mathematical behavior nor the operations count compared to A2/A3, but it allows to use A4 more efficiently. CS_0 and CS_1 operations also act as “bridge” to exchange data between the 2 clusters when shared into a single CSWAP-V3 unit as illustrated in Fig. 4.

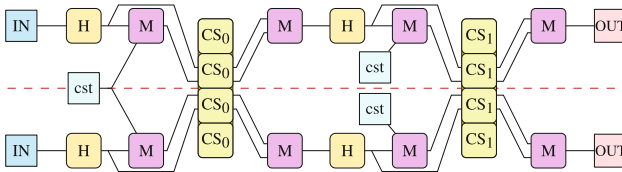


Fig. 5. Modified x DBLADD formula for architecture A4 with the new CSWAP behavior.

Table 9. FPGA implementation results for architecture A4 (all BRAMs are 18 Kb ones, only 17×17 multipliers were used in DSP slices for all FPGAs).

FPGA	w [bit]	LUT	FF	Logic slices	DSP blocks	RAM blocks	Freq. [MHz]	Clock cycles	Time [ms]
V4	34	1695	2950	2158	22	7	324	142,119	0.44
	68	2804	4282	3184	22	9	290	128,021	0.44
	136	3171	4994	3337	22	13	299	125,456	0.42
V5	34	1370	2953	1013	22	7	358	142,119	0.40
	68	2095	4259	1358	22	9	337	128,021	0.38
	136	2514	4952	1589	22	13	313	125,456	0.40
S6	34	1564	2089	758	22	7	262	142,119	0.54
	68	2387	4030	1060	22	9	239	128,021	0.54
	136	3181	4786	1136	22	13	251	125,456	0.50

Architecture A4 (Fig. 4) is based on two identical clusters. Each cluster contains: 1 `AddSub`, 1 `Mult`, 1 data memory and 1 local communication system. The shared `CSWAP-V3` unit uses a 2-bit control mode to select the relevant swapping pattern according to the dependency graph in Fig. 5. The control is shared for the 2 clusters, management of ML iterations and external inputs/outputs (at beginning/end of scalar multiplication).

Complete implementation results for A4 are reported in Table 9. A few elements are summarized below:

- Clock cycles count is reduced for `w34` but slightly increased for `w68` and `w136` compared to A3. This is due to additional constraints on the scheduler for A4 since all units do not share a common memory and clusters can only exchange data during the new modified `CSWAP-V3`.
- Frequency is higher in A4 compared to A3 in most of case (and very similar in the other cases). This is due to a more local control and a smaller fanout (most of signals are local to each cluster).
- Computation time is significantly reduced for small w values. For instance, `w34` leads to a similar speed than A3 but with much smaller architecture.
- On V5, the fastest solution is the intermediate configuration `w68`.
- DSP slices amount is exactly the same in A4 and A3 (both use 2 `Mult` units).
- BRAMs amount is larger in A4 than A3 due to the 2 local memories (one in each cluster). The number of BRAMs increases with w accordingly to Table 2 configurations.
- Area results in terms of LUTs are quite different from previous architectures. It increases for `w34` (up to +15%) but decreases for `w136` (up to -16%) compared to A3.
- Adding a second memory unit allows parallel read/write accesses and helps to quickly extract more parallel operations.

As a conclusion for architecture A4, selecting the right set of parameters depends a lot on the objective (high speed or low cost) and the FPGA family. When the main objective is selecting the absolute smallest accelerator, A4 is less interesting than A3. When the main objective is selecting the absolute fastest accelerator, A4 is interesting for low-cost S6 FPGA (but A3 is better for V4/V5).

But in practice, A4 is interesting for accelerators almost as fast as the fastest A3 but for much smaller area. For instance on V4, w34 configuration is only 5% slower than the absolute fastest A3 solution with an area reduced by 55% in LUTs and 22% in BRAMs.

6 Comparisons

Figure 6 reports all the implementation results for the 4 proposed architectures and w configurations on 3 different FPGAs. This figure only reports LUTs for area since there are only a very few different numbers of BRAMs and DSP slices given in Table 10. The best configuration (architecture type, w) depends on the objective (high speed or low area) and target FPGA. Then exploration tools at architecture level are helpful for designers and users. For low-area solution, A2 is always the best one. For high speed, A4 is a very good cost-performance trade-off on V4 and S6 (A3 is the fastest on V5 but with a large area).

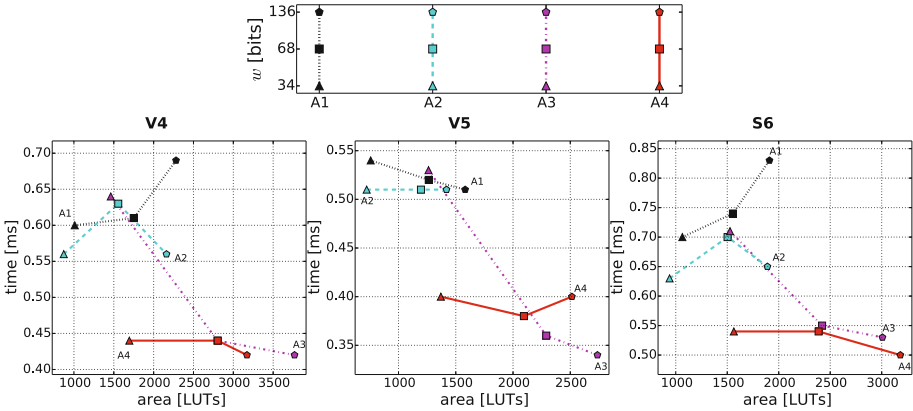


Fig. 6. Trade-offs for our architectures A1–4 in terms of area (LUTs) and computation time for all w configurations and FPGAs (legend is top figure).

Most of hardware HECC implementations use curves over \mathbb{F}_{2^n} with low security levels (typically 81–89 bits fields). Table 11 reports some of them. None of those HECC implementations embed hardware level protections against SCAs. Some of them use algorithmic protections for scalar multiplication such as the Montgomery ladder. Our \mathbb{F}_p accelerators show similar computation times but for a much higher security level (128 bits) on more recent FPGAs. To the best of our knowledge, we found only one hardware implementation of HECC over \mathbb{F}_p in [1]. It is an $0.13\ \mu\text{m}$ ASIC implementation for 81-bit generic prime p with 502.8 ms computation time for scalar multiplication at 1 MHz. The very low reported frequency makes comparisons quite difficult.

Directly comparing our accelerators with implementations of HECC over \mathbb{F}_{2^n} for much lower security level is not possible. Then, in Table 12, we report some of

Table 10. Summary of our most interesting FPGA implementation results (those on Pareto front from Fig. 6).

Archi.	w [bit]	Target	Logic slices	DSP blocks	RAM blocks	Freq. [MHz]	Time [ms]
A2	34	V4	1121	11	4	330	0.56
A3	136		3660	22	9	285	0.42
A4	34		2158	22	7	324	0.44
A2	34	V5	541	11	4	360	0.51
A3	136		1594	22	9	348	0.34
A4	34		1013	22	7	358	0.40
A2	34	S6	381	11	4	293	0.63
A3	136		1131	22	9	225	0.53
A4	34		758	22	7	262	0.54

Table 11. FPGA implementation results for various HECC solutions over \mathbb{F}_{2^n} from state of the art (warning: security levels are much lower than our solutions). For \mathbb{F}_{2^n} DSP slices cannot be used. Values with a “*” are estimated number of RAM blocks based on paper explanations.

Ref.	Year	Target	n	LUT	FF	Logic slices	RAM blocks	Freq. [MHz]	Time [ms]
[3]	2006	Virtex 2 Pro	83	20999	n.a.	11296	n.a.	166	0.5
[8]	2008	XC2V4000	83	n.a.	n.a.	2316	6	125	0.31
[13]	2004	XC2V4000	89	8451	2178	4995	1	54	1.02
		XC2V4000	89	16459	4437	9950	0	57	0.44
[25]	2006	Virtex 2 Pro	83	n.a.	n.a.	2446	1*	100	0.99
		Virtex 2 Pro	83	n.a.	n.a.	6586	3*	100	0.42
[26]	2016	Virtex 2	83	n.a.	n.a.	5734	n.a.	145	0.3
		XC5V240	83	n.a.	n.a.	5086	n.a.	175	0.29
[27]	2004	Virtex 2 Pro	81	n.a.	n.a.	4039	1	57	0.79
		Virtex 2 Pro	81	n.a.	n.a.	7737	0	61	0.39
		XC2V4000	81	n.a.	n.a.	3955	1	54	0.83
		XC2V4000	81	n.a.	n.a.	7785	n.a.	57	0.42
[7]	2007	XC2V8000	113	n.a.	n.a.	25271	n.a.	45	2.03

the best FPGA implementations results we found in the state of the art for ECC solutions over \mathbb{F}_p and 128 bits security level. In practice using hundred of DSP blocks and BRAMs may not be a realistic solution for embedded systems. In [2] several SCAs protections have been presented: DBL&ADD-always, ML, scalar randomization, units with uniform behavior, randomization of memory addresses and noise addition. Those protections impact the number of logic slices but not those of DSPs and BRAMs (very huge is this work).

Compared to [19], a very optimized fast and compact solution from state of the art using randomized Jacobian coordinates, our accelerator have a very similar computation time (0.44ms) but with 40% reduction in DSP and RAM blocks and 53% reduction of logic slices on V4 FPGA (similar for V5).

Table 12. FPGA implementation results for various ECC solutions over \mathbb{F}_p and 128-bit security level from state of the art.

Ref.	Year	Target	p	LUT	FF	Logic slices	DSP blocks	RAM blocks	Freq. [MHz]	Time [ms]
[2]	2014	XCV6FX760	NIST-256	32900	n.a.	11200	289	128	100	0.4
[11]	2008	XC4VFX12	NIST-256	2589	2028	1715	32	11	490	0.5
		XC4VFX12	NIST-256	34896	32430	24574	512	176	375	0.04
[17]	2012	XC4VFX12	GEN-256	n.a.	n.a.	2901	14	n.a.	227	1.09
		XC5VLX110	GEN-256	n.a.	n.a.	3657	10	n.a.	263	0.86
[19]	2013	XC4VLX100	GEN-256	5740	4876	4655	37	11	250	0.44
		XC5LX110T	GEN-256	4177	4792	1725	37	10	291	0.38

7 Conclusion and Future Prospects

We proposed the first hardware implementation of Kummer based HECC solution for 128-bit security level. Various architectures and parameters have been explored using in-house tools. Several architectures with different amount of internal parallelism have been optimized and fully implemented on 3 different FPGAs. The obtained results lead to similar speed than the best curve based solutions for embedded systems but with an area almost divided by 2 (-40% for DSP and RAM blocks and -60% for logic slices). Those results were obtained with generic prime fields and fully programmable architectures (which is not the case in most of state of the art implementations).

In the future, we plan to optimize our tools and architectures, evaluate the security against SCAs using real measurement setup, automate the control generation of the accelerator, and publish our architectures as open source hardware.

Acknowledgment. This work was done in the HAH project <http://h-a-h.inria.fr/> partially funded by Labex CominLab, Labex Lebesgue and Brittany Region.

References

1. Ahmadi, H.-R., Afzali-Kusha, A., Pedram, M., Mosaffa, M.: Flexible prime-field genus 2 hyperelliptic curve cryptography processor with low power consumption and uniform power draw. *ETRI J.* **37**(1), 107–117 (2015)
2. Alrimeih, H., Rakhmatov, D.: Fast and flexible hardware support for ECC over multiple standard prime fields. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **22**(12), 2661–2674 (2014)
3. Batina, L., Mentens, N., Preneel, B., Verbauwhede, I.: Flexible hardware architectures for curve-based cryptography. In: *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4839–4842. IEEE, May 2006
4. Bernstein, D.J., Lange, T.: Explicit-formulas database. <http://hyperelliptic.org/EFD/>
5. Bos, J.W., Costello, C., Hisil, H., Lauter, K.: Fast cryptography in genus 2. *J. Cryptol.* **29**(1), 28–60 (2016)
6. Cohen, H., Frey, G. (eds.): *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete Maths and Applications. Chapman & Hall/CRC, London (2005)

7. Elias, G., Miri, A., Yeap, T.-H.: On efficient implementation of FPGA-based hyperelliptic curve cryptosystems. *Comput. Electr. Eng.* **33**(5), 349–366 (2007)
8. Fan, J., Batina, L., Verbaauwhede, I.: HECC goes embedded: an area-efficient implementation of HECC. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 387–400. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_25
9. Gallin, G., Tisserand, A.: Hyper-threaded multiplier for HECC. In: Proceedings of 51st Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA. IEEE, October 2017
10. Gaudry, P.: Fast genus 2 arithmetic based on theta functions. *J. Math. Cryptol.* **1**(3), 243–265 (2007)
11. Güneysu, T., Paar, C.: Ultra high performance ECC over NIST primes on commercial FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 62–78. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_5
12. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004). <https://doi.org/10.1007/b97644>
13. Kim, H.W., Wollinger, T., Choi, Y.J., Chung, K.I., Paar, C.: Hyperelliptic curve coprocessors on a FPGA. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 360–374. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31815-6_29
14. Koblitz, N.: Hyperelliptic cryptosystems. *J. Cryptol.* **1**(3), 139–150 (1989)
15. Koc, C.K., Acar, T., Kaliski, B.S.: Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro* **16**(3), 26–33 (1996)
16. Satoh Laboratory and Morita Tech: Side-channel attack user reference architecture (SAKURA) (2013)
17. Lai, J.-Y., Wang, Y.-S., Huang, C.-T.: High-performance architecture for elliptic curve cryptography over prime fields on FPGAs. *Interdiscip. Inf. Sci.* **18**(2), 167–173 (2012)
18. Lange, T.: Formulae for arithmetic on genus 2 hyperelliptic curves. *Appl. Algebra Eng. Commun. Comput.* **15**(5), 295–328 (2005)
19. Ma, Y., Liu, Z., Pan, W., Jing, J.: A high-speed elliptic curve cryptographic processor for generic curves over $\text{GF}(p)$. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 421–437. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_21
20. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, Heidelberg (2007). <https://doi.org/10.1007/978-0-387-38162-6>
21. Montgomery, P.L.: Modular multiplication without trial division. *Math. Comput.* **44**(170), 519–521 (1985)
22. Montgomery, P.L.: Speeding the pollard and elliptic curves methods of factorisation. *Math. Comput.* **48**(177), 243–264 (1987)
23. Orup, H.: Simplifying quotient determination in high-radix modular multiplication. In: Proceedings of 12th IEEE Symposium on Computer Arithmetic (ARITH), pp. 193–199, Bath, UK. IEEE, July 1995
24. Renes, J., Schwabe, P., Smith, B., Batina, L.: μ Kummer: efficient hyperelliptic signatures and key exchange on microcontrollers. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 301–320. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_15

25. Sakiyama, K., Batina, L., Preneel, B., Verbauwhede, I.: Superscalar coprocessor for high-speed curve-based cryptography. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 415–429. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_33
26. Sghaier, A., Massoud, C., Zeghid, M., Machhout, M.: Flexible hardware implementation of hyperelliptic curves cryptosystem. *Int. J. Comput. Sci. Inf. Secur. (IJCSIS)* **14**(4), 1–7 (2016)
27. Wollinger, T.: Software and hardware implementation of hyperelliptic curve cryptosystems. Ruhr University Bochum (2004)

Bricklayer Attack: A Side-Channel Analysis on the ChaCha Quarter Round

Alexandre Adomnicai^{1,3}, Jacques J. A. Fournier²,
and Laurent Masson¹

¹ Trusted Objects, Aix-en-Provence, France
{a.adomnicai,l.masson}@trusted-objects.com

² CEA-Leti, Grenoble, France

jacques.fournier@cea.fr

³ EMSE, Gardanne, France

Abstract. ChaCha is a family of stream ciphers that are very efficient on constrained platforms. In this paper, we present electromagnetic side-channel analyses for two different software implementations of ChaCha20 on a 32-bit architecture: one compiled and another one directly written in assembly. On the device under test, practical experiments show that they have different levels of resistance to side-channel attacks. For the most leakage-resilient implementation, an analysis of the *whole* quarter round is required. To overcome this complication, we introduce an optimized attack based on a divide-and-conquer strategy named *bricklayer attack*.

Keywords: ChaCha · Implementation · Side-channel attacks

1 Introduction

ChaCha [7] is a family of stream ciphers introduced by Daniel J. Bernstein in 2008. It is a variant of the Salsa20 family [8], which is part of the eSTREAM portfolio [4], providing better diffusion for similar performances. ChaCha is an ARX-based cipher, which means that it only uses modular additions, rotations and bitwise XORs. It has been widely adopted for encryption, as well as for random number generation in many operating systems (*e.g.* Linux, OpenBSD) and protocols (*e.g.* SSH, TLS). Moreover, the upcoming version 1.3 of the Transport Layer Security (TLS) protocol [35] will allow Authenticated Encryption with Associated Data (AEAD) cipher suites only, leaving AES-CCM [31], AES-GCM [37] and ChaCha20-Poly1305 [25] as the only three options. This update should significantly increase the use of ChaCha in the near future. On top of that, the Internet of Things (IoT) should be in favour of the ChaCha deployment (*e.g.* Apple HomeKit for IoT devices [2]), since its instances are cheaper than AES on microcontrollers that do not have any dedicated cryptographic hardware. For instance, on Android phones, HTTPS connections from Chrome browsers to Google now use ChaCha20-Poly1305 [12].

As a result of its standardization, ChaCha is under close scrutiny with regards to cryptanalysis, especially regarding differential attacks [3, 14, 28, 38, 40]. Recently, studies have been carried out to evaluate its physical security, especially regarding fault attacks [24, 32]. However, only one side-channel analysis has been proposed so far [21]. We believe that further work must be undertaken in this field since ChaCha is particularly well suited for embedded devices.

Our Contribution. In this paper, we focus on the side-channel analysis of ChaCha by taking two different implementations into consideration.

First, we investigate the OpenSSL C source code compiled on a 32-bit ARM microcontroller. It results in a straightforward attack path, which consists in targeting each 32-bit key word independently.

The second target is an assembly implementation which saves some memory accesses. We highlight that, on the device under test (DUT), this slight modification protects from the only side-channel attack published to date. Nevertheless, our implementation remains vulnerable even though attack paths are more complex. We tackle this problem by introducing the *bricklayer attack*, which is based on a divide-and-conquer approach, and emphasize that attacking from the keystream rather than from the input is way more efficient.

Outline. First, we present the ChaCha family of stream ciphers before providing an outline of side-channel attacks. Then, we describe our approaches on performing electromagnetic analyses depending on software implementations of ChaCha. Subsequently, we present our practical results and discuss the feasibility of conducting these attacks in real-world scenarios. Finally, we analyze the overhead introduced by the masking countermeasure in the specific case of ChaCha20.

2 The ChaCha Family of Stream Cipher

As its predecessor, and unlike traditional stream ciphers, ChaCha does not have an initialization phase since it works like a block cipher used in counter (CTR) mode [18]. Its core is an ARX-based function which maps a 512-bit input block to a 512-bit output key stream. Input blocks are built by arranging data in a 4×4 matrix where each element is a 32-bit word. The encryption key fills half of the matrix as it is 256-bit long, while the two remaining quarters are respectively occupied by the inputs and the constant ‘expand 32-byte k’. This constant aims at reducing the amount of data an attacker can control while the inputs refer to a nonce which is built from the block counter and the initial vector (IV) (Fig. 1).

The core function is defined by iterating several rounds on the input block, where each round consists of four parallel quarter round (QR) operations. A QR updates 4 words (*i.e.* a block quarter) as defined in Algorithm 1 where \boxplus means addition modulo 2^{32} , \oplus means XOR and \lll means left bitwise rotation.

Depending on the round number (enumerated from 0), each QR operates either on a column, or on a diagonal. ChaChaR refers to a specific instance

‘expa’	‘nd 3’	‘2-by’	‘te k’
k_0	k_1	k_2	k_3
k_4	k_5	k_6	k_7
nonce ₀	nonce ₁	nonce ₂	nonce ₃

Fig. 1. ChaCha’s input block initialization

Algorithm 1. ChaCha quarterround(a , b , c , d)

$a \boxplus= b;$	$d \oplus= a;$	$d \lll= 16;$
$c \boxplus= d;$	$b \oplus= c;$	$b \lll= 12;$
$a \boxplus= b;$	$d \oplus= a;$	$d \lll= 8;$
$c \boxplus= d;$	$b \oplus= c;$	$b \lll= 7;$

where R rounds are used. Several variants are defined with 8, 12 or 20 rounds, defining different trade-offs between security and performance. Recently, it has been shown under certain assumptions that ChaCha12 is sufficiently secure to ensure a 256-bit security level [14]. Nevertheless, ChaCha20 remains the most widespread instance for security margins. In many implementations, ChaCha R uses $\frac{R}{2}$ iterations of double rounds instead of R rounds, which consists in a column round and a diagonal one.



On top of iterating several rounds on the input block, an additional step is required. The reason is that while QRs scramble blocks beyond recognition, they are invertible. Therefore, applying the reverse of each operation in the reverse order leads to the original block and thus, the encryption key. ChaCha prevents this by adding the original block to the scrambled one, word by word, in order to generate the pseudo-random block. The whole encryption process is detailed in Algorithm 2.

3 Background on Side-Channel Attacks

3.1 Correlation Electromagnetic Analysis

Cryptographic primitives are usually built to resist to mathematical cryptanalysis or exhaustive key search. However, they are designed to be finally executed

Algorithm 2. ChaChaR encryption

Require:

n -bit plaintext P
 encryption key k
 counter ctr
 IV iv

Ensure: n -bit ciphertext C **for** i from 0 to $\lfloor n/512 \rfloor$ **do** $B \leftarrow \text{init}(k, ctr, iv)$ ▷ input block initialization $B' \leftarrow B$ ▷ working variable**for** j from 0 to $\frac{R}{2} - 1$ **do**quarterround($B'_0, B'_4, B'_8, B'_{12}$) ▷ column roundsquarterround($B'_1, B'_5, B'_9, B'_{13}$)quarterround($B'_2, B'_6, B'_{10}, B'_{14}$)quarterround($B'_3, B'_7, B'_{11}, B'_{15}$)quarterround($B'_0, B'_5, B'_{10}, B'_{15}$) ▷ diagonal roundsquarterround($B'_1, B'_6, B'_{11}, B'_{12}$)quarterround($B'_2, B'_7, B'_8, B'_{13}$)quarterround($B'_3, B'_4, B'_9, B'_{14}$)**end for** $B \leftarrow B \boxplus B'$ ▷ final block addition $C_i \leftarrow P_i \oplus B$ $ctr \leftarrow ctr + 1$ **end for**

on a given processor with its own physical characteristics. Electronic circuits are inherently leaky as they produce emissions that make it possible for an attacker to deduce how the circuit works and what data is being processed. Because these emissions are nothing more than side effects, their use to recover cryptographic keys has been termed ‘side-channel attacks’. Since the publication of Differential Power Analysis (DPA) [23], it is common knowledge that the analysis of the power consumed by the execution of a cryptographic primitive might reveal information about the secret involved.

A few years later, Correlation Power Analysis (CPA) has been widely adopted over DPA as it requires fewer traces and has been shown to be more efficient [11]. The principle is to target a sensitive intermediate state of the algorithm and try to predict its value from the known input and different key guesses. Then, to uncover the link between these predictions and the leakage measurements, the Pearson correlation coefficient between these two variables is computed using an appropriate leakage model. The Hamming weight (HW) and the Hamming distance (HD) model are the most commonly used models to simulate the leakage of a cryptographic device. For each key hypothesis, it results in a value between -1 (total negative correlation) and 1 (total positive correlation) for every point in time, indicating how much the prediction correlates with the recorded values over several measurements. The formula of this coefficient is

$$\text{Corr}(X, Y) = \frac{E(X \cdot Y) - E(X) \cdot E(Y)}{\sqrt{E((X - E(X))^2) \cdot E((Y - E(Y))^2)}} \quad (1)$$

where $E(X)$ is the expected value of the random variable X . Finally, the hypothesis which matches with the real key should return a significantly higher coefficient than the other hypotheses. This attack remains valid when analyzing electromagnetic emanations [19,34] instead of power consumption, since they are mainly due to the displacement of current through the rails of the metal layers. In this case, we talk about Correlation ElectroMagnetic Analysis (CEMA).

3.2 Selection Function

The intermediate state y on which the side-channel attack focuses is defined by a *selection function* $\varphi(x, k) = y$, which is part of the encryption algorithm. It depends on x , a known part of the input and on k , an unknown part of the secret key. Usually, selection functions are chosen to be easy to compute, typically at the beginning of the encryption or decryption process. Furthermore, a valuable property for selection functions is high non-linearity as it ensures a good distinguishability between the correct and incorrect key guesses. Indeed, correlation between the leakage and the prediction will be close to zero if the key guess is incorrect due to their non-linear relationship.

In case of ARX structures, the non-linearity only relies on modular additions, while diffusion is provided by rotations (diffusion within single words) and XORs (diffusion between words). Although the carry propagation in the modular addition results in some non-linearity, it is not as good a candidate as S-boxes. It can be explained by the fact that most significant bits in the output of a modular addition are more subject to non-linearity than least significant ones. However, side channel attacks remain possible as shown in numerous publications [10,26,41].

4 Side-Channel Overview of ChaCha

4.1 ChaCha Case Study

To set up such a side-channel attack, one has to determine an attack path (*i.e.* to choose a selection function) either starting from the plaintext, or from the ciphertext. Physical attacks against stream ciphers can be challenging because the key stream is computed independently from the plaintext/ciphertext, which interferes in the relationship between known values and the secret key. However, from a side-channel point of view, ChaCha differs significantly from other stream ciphers' designs such as linear-feedback shift registers where the key is only directly involved during registers' initialization. Indeed, as ChaCha operates like a block cipher in CTR mode, the key is directly manipulated everytime a 512-bit block needs to be encrypted. More precisely, each key word directly interacts

with other data during the first round (after which they have been updated) and again during the final block addition.

An attack that takes advantage of the first round has already been published in [21]. The attack on the i^{th} column round ($0 < i < 4$) relies on the selection function defined by

$$\varphi_0(\text{nonce}_i, \tilde{k}_i \parallel k_{i+4}) = \left((\text{nonce}_i \oplus \tilde{k}_i) \lll 16 \right) \boxplus k_{i+4} \quad (2)$$

where $\tilde{k}_i = k_i \boxplus \text{constant}_i$. However, this selection function forces the attacker to target two key words at once, which results in a key search space $|\mathcal{K}| = 2^{64}$. Since the bit-size of the targeted subkey determines the memory complexity of the side-channel attack, one can understand why this would be undoable in practice. To get around this problem, the authors exploit the QRs' intermediate states in order to operate step by step. They propose to first recover k_i by targeting $\text{nonce}_i \oplus \tilde{k}_i$ and then take advantage of its knowledge to find k_{i+4} . Therefore, recovering k_i and k_{i+4} requires the knowledge of nonce_i . However, the paper also describes an attack path that allows to recover the entire key with the knowledge of only two words. This latter exploits several intermediate states in the first two rounds.

Regarding the final block addition, an attacker could choose $\varphi(x, k) = x \boxminus k$ where x refers to a keystream word and \boxminus refers to modular subtraction. Compared to the previous attack path, it has the advantage of recovering all key words using the modular subtraction as selection function. Moreover, all keystream words are pseudorandom values, which is not necessarily the case for nonces. However, this selection function requires the knowledge of the keystream (*i.e.* both plaintext and ciphertext).

Throughout this paper, we will make the assumption that an attacker has access to all this information. In Sect. 6 we discuss the attacks' feasibility in practice and thus, whether our assumptions are reasonable.

4.2 Implementation Aspects

When targeting software implementations on load/store architectures, data transfers due to memory accesses (*i.e.* loads and stores between memory and registers) are known to leak the most information compared to arithmetic and logic operations [13, 30], which only occur between registers and are usually unexploitable in practice [9]. Our practical experiments on the DUT presented in Sect. 6 verified this hypothesis. Therefore, the intermediate values that are manipulated by these sensitive operations should be easiest to target, introducing a direct link between selection functions and implementation aspects.

Throughout this paper we will study selection functions in relation to memory accesses, assuming they are the main source of exploitable leakage.

4.3 OpenSSL Implementation

First, we decided to attack a C implementation of ChaCha20 in order to see how compilers can deal with ARX structures and memory accesses. To do so,

we compiled the ChaCha20 C implementation from OpenSSL (version 1.0.1f) for an ARM Cortex-M3 microcontroller using the GNU ARM C compiler 5.06 (update 2). Regardless of the optimization level chosen (from `-O0` to `-O3`), within a QR, each addition and each rotation is followed by a STR instruction. Hence, these memory accesses allowed us to carry out the attacks described above. Practical results are briefly presented in Sect. 6 for comparative purposes.

4.4 Side-Channel Analysis of the Salsa20 Quarter Round

In the next section, we show how memory accesses can be easily managed to remove the leakage of intermediate states within a QR. This implies to target the QR output without taking its intermediate values into consideration, making the attacks presented in [21] irrelevant in this case. Although such an analysis has already been performed on Salsa20 [29], it does not apply to ChaCha.

Algorithm 3. Salsa20 quarterround(a , b , c , d)

$b \oplus = (a \boxplus d) \lll 7;$	$c \oplus = (b \boxplus a) \lll 9;$
$d \oplus = (c \boxplus b) \lll 13;$	$a \oplus = (d \boxplus c) \lll 18;$

In the case of Salsa20, as described in Algorithm 3, the update of the second input only depends on itself and two others (the first and the last). This allows to recover the key words involved in this computation as first/last input words, with two other ‘non-key’ operands (*i.e.* constant and nonce). The attack consists in performing a CPA on a 32-bit value using a divide-and-conquer (D&C) approach, which consists in separating the attack into $\lceil \frac{32}{n} \rceil$ computations on n -bit windows in parallel. The other key words that do not match these requirements were retrieved by using the knowledge of those which have been previously recovered. This allowed to keep a search space of 2^{32} instead of 2^{64} . On top of providing better diffusion, the ChaCha QR gives *each* input word a chance to affect the other three twice. This adjustment makes the attack irrelevant against ChaCha since the key search space cannot be less than 2^{64} in any case.

5 Side-Channel Analysis of the Quarter Round

Throughout this section, for greater clarity, we assume that all operators are left-associative so that

$$a \boxplus b \oplus c \lll d \iff (((a \boxplus b) \oplus c) \lll d).$$

5.1 Optimizing Memory Accesses

A solution to overcome attacks on intermediate states within QRs is a straightforward assembly implementation, which is a good way to reduce memory access instructions for load/store architectures. As explained in [9], for some instances of ARX lightweight block ciphers like Simon and Speck [5], it is possible to keep the whole state in registers during the entire encryption process. Thereby, they can be implemented in assembly without having to execute a single STR instruction during the whole encryption process, drastically reducing the amount of leakage.

Unfortunately, in the case of ChaCha, the state consists of 16 32-bit words. Therefore, it would require a 32-bit CPU with at least 16 general-purpose registers (excluding the stack pointer, the program counter and other specific cases such as hardwired registers) to avoid memory accesses. As our chip only has 13 general-purpose registers, we implemented ChaCha so that word values are loaded into registers at the beginning of each QR and are then stored in RAM at the end. Furthermore, during the last round, related key words are also loaded into registers at the beginning of QRs, resulting in

$$\begin{aligned} & \text{quarterround}'(x_0, x_5, x_{10}, x_{15}, k_1, k_6) \\ & \text{quarterround}'(x_1, x_6, x_{11}, x_{12}, k_2, k_7) \\ & \text{quarterround}'(x_2, x_7, x_8, x_{13}, k_3, k_4) \\ & \text{quarterround}'(x_3, x_4, x_9, x_{14}, k_0, k_5) \end{aligned}$$

where $\text{quarterround}'(a, b, c, d, x, y) = \text{quarterround}(a, b, c, d) \boxplus (0, x, y, 0)$. This method protects against leakages that would allow an attack from the keystream using the modular subtraction as selection function. Thus, these elementary implementation tricks imply to analyze the side-channel resilience of the *whole* QR.

5.2 Focusing on the Quarter Round

As every word influences the three others, and is updated twice, the simplest selection function would be defined by focusing, during the first column rounds, on the word which is completely updated at first, resulting in having

$$\varphi_1(\text{nonce}_i, k_i \parallel k_{i+4}) = \text{nonce}_i \oplus \tilde{k}_i \lll 16 \boxplus k_{i+4} \oplus k_i \lll 12 \boxplus \tilde{k}_i. \quad (3)$$

However, as previously mentioned, this implies a side-channel attack on 64 bits, which is not feasible in practice. Therefore, we investigated the relevance of the D&C approach in this specific case. Figure 2 sketches how key words are involved in computations. It results that targeting n bits of $y = \varphi_1(\text{nonce}_i, k_i \parallel k_{i+4})$ does not lead to a complexity equal to 2^{2n} since rotations make different n -bit windows interact with each other. As there is a rotation of 16 bits followed by another one of 12, some bits of \tilde{k}_i may overlap. Hence, the key search space depends on

the windows' size.

$$|\mathcal{K}| = \begin{cases} 2^{4n}, & \text{if } n \leq 4 \\ 2^{3n+4}, & \text{if } 4 \leq n \leq 12 \\ 2^{2n+16}, & \text{if } 13 \leq n \leq 16 \\ 2^{n+32}, & \text{otherwise} \end{cases} \quad (4)$$

Furthermore, rotations are discarded from the selection function, resulting in

$$\varphi_{2,n} \left(\text{nonce}_i, \tilde{k}_i^A \parallel k_i^B \parallel k_{i+4}^B \parallel \tilde{k}_i^C \right) = \text{nonce}_i^A \oplus \tilde{k}_i^A \boxplus_n k_{i+4}^B \oplus k_i^B \boxplus_n \tilde{k}_i^C \quad (5)$$

where superscripts refer to intervals that define n -bit windows.

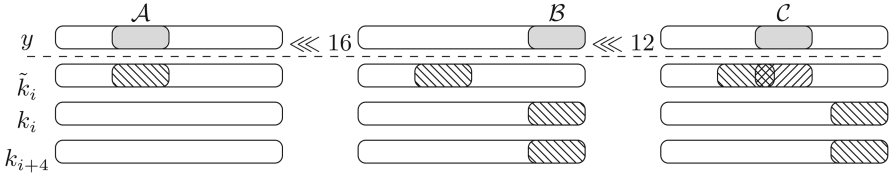


Fig. 2. D&C approach on the ChaCha QR, $n = 8$

In order to evaluate this method, we performed software simulations using the HW model (without any additional noise) and random nonces. As expected, the right key matches with the highest correlation coefficient. Nevertheless, some other hypotheses also lead to the maximum coefficient as shown in Fig. 3, resulting in collisions.

Definition 1 (Collision). Let $\varphi(n, k)$ be a selection function and κ be the right key hypothesis. A collision is an hypothesis κ' such that $\varphi(n, \kappa) = \varphi(n, \kappa')$ for all n .

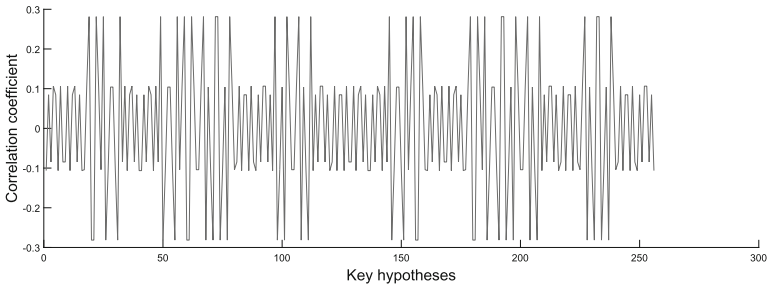


Fig. 3. Attack simulation on $\varphi_{2,2}$

Proposition 1. *An attack on $\varphi_{2,n}$ returns up to $n \cdot 2^{n+2}$ collisions.*

Another point that has not been discussed so far is the drawback caused by carry propagations. Except when focusing on the least significant bits (LSBs), one has no way of knowing if subkeys involved in additions are affected by a carry. Thus, the positions of targeted windows are very important. Plus, we made the choice to dissociate \tilde{k}_i from k_i in order to prevent from erroneous predictions of $k_i^A \boxplus_n \text{constant}_i^A$ and $k_i^C \boxplus_n \text{constant}_i^C$. For instance, in Fig. 2, \tilde{k}_i^C is the only hypothesis which could be erroneous due to a carry propagation on its addend. As a result, an attacker should mount one attack taking this carry into consideration, and another one without. This would mean that the total number of collisions would be doubled. Although this selection function may provide some information, we chose to investigate a more efficient attack path.

5.3 Benefits of the Reverse Function

The ChaCha QR is trivially invertible and the inverse quarter round (IQR) is defined in Algorithm 4.

Algorithm 4. ChaCha inv_quarterround(a, b, c, d)

$b \ggg 7;$	$b \oplus= c;$	$c \boxminus= d;$
$d \ggg 8;$	$d \oplus= a;$	$a \boxminus= b;$
$b \ggg 12;$	$b \oplus= c;$	$c \boxminus= d;$
$d \ggg 16;$	$d \oplus= a;$	$a \boxminus= b;$

What matters here is that each input word does not have a chance to influence the other three, since the first word does not impact the update of the second one. Hence, the overall selection can be defined as below

$$\varphi_3(b \parallel c \parallel \tilde{d}_i, k_b \parallel k_c) = (b \boxminus k_b \ggg 7) \oplus (c \boxminus k_c \ggg 12) \oplus (c \boxminus k_c \boxminus \tilde{d}_i) \quad (6)$$

where $\tilde{d}_i = d_i \boxminus \text{nonce}_i$. Regarding the D&C approach where rotations are discarded, it results in the following selection function.

$$\varphi_{4,n}(b \parallel c \parallel \tilde{d}_i, k_b^A \parallel k_c^B \parallel k_c^C) = (b^A \boxminus_n k_b^A) \oplus (c^B \boxminus_n k_c^B) \oplus (c^C \boxminus_n k_c^C \boxminus_n \tilde{d}_i^C) \quad (7)$$

As less words are involved, the key search space is reduced and still depends on the windows' size.

$$|\mathcal{K}| = \begin{cases} 2^{3n}, & \text{if } n \leq 12 \\ 2^{2n+12}, & \text{if } 12 \leq n \leq 20 \\ 2^{n+32}, & \text{otherwise} \end{cases} \quad (8)$$

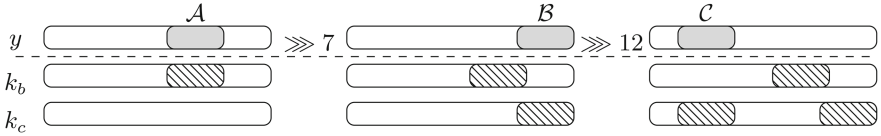


Fig. 4. D&C approach on the ChaCha IQR, $n = 8$

However, since the rotations are less pronounced, key words do not overlap if the windows' size does not exceed 12 bits, as depicted in Fig. 4. Throughout the rest of this section, we only consider the case where $n \leq 12$.

As before, key hypotheses might be affected by carry propagations. However, another advantage of $\varphi_{4,n}$ over $\varphi_{2,n}$ is that one knows the *entire* 32-bit minuend (*i.e.* b or c). Thus, depending on its value, one can calculate the probability of a carry propagation. For instance, when targeting $k_b^{[x,x+n]}$, the probability is

$$p = \mathbb{P}\left(k_b^{[0,x]} > b^{[0,x]}\right) = \frac{2^x - (b^{[0,x]} + 1)}{2^x}. \quad (9)$$

For our simulation with $n = 4$, we took a carry into consideration only if $p > 0.75$. On top of providing a smaller key search space, $\varphi_{4,n}$ is less prone to collisions as shown by our simulation depicted in Fig. 5.

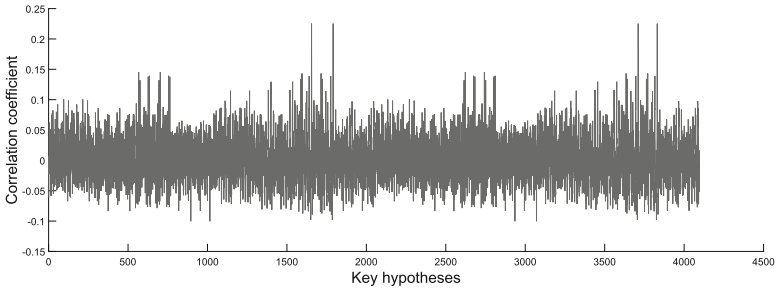


Fig. 5. Attack simulation on $\varphi_{4,4}$

Proposition 2. *An attack on $\varphi_{4,n}$ returns 4 collisions.*

Proof. Flipping the MSB of the minuend/subtrahend also flips the MSB of the modular difference. Therefore, in the case of $\varphi_{4,n}$, flipping the MSB of two n -bit key windows leads to the same output. As a result, the number of collisions is equal to $1 + \binom{3}{2} = 4$. \square

This property allows to halve the key search space (*i.e.* $|\mathcal{K}| = 2^{3n-1}$), since all collisions can be retrieved from just one. In the next section, we suggest a more efficient method than repeating this computation over several windows and then sorting the right key from the collisions.

5.4 Overview of the Brickerlayer Attack

Once collisions have been found using $\varphi_{2,n}$ or $\varphi_{4,n}$, one has to reiterate the same procedure on different windows. Instead of executing several attacks in parallel, we suggest to take advantage of windows that have been previously recovered, in order to target larger ones. For instance, once 4 collisions have been found after an attack on $\varphi_{4,n}$, one can target $\varphi_{4,m}$, where $m > n$, with a complexity $|\mathcal{K}| = 2^{3(m-n)+1}$.

Proceeding in this sequential manner has two advantages. First, taking the carry propagation into consideration is only necessary during the first attack. This property is especially interesting for $\varphi_{2,n}$ since there is no way to estimate carry propagations in this case. Second, each attack cancels collisions from the previous ones, since the positions of the collision bits are changed. For instance, regarding $\varphi_{4,n}$ where collisions only depend on MSBs, the bricklayer approach transforms previous collisions into the predictions' lower bits, allowing the correct collision to stand out. This property is less efficient in the case of $\varphi_{2,n}$ since collisions depends on all bits of the n -bit word. Therefore, the correct collision does not stand out directly but some wrong hypotheses are still discarded.

An example application of the bricklayer attack using $\varphi_{4,n}$ is depicted in Fig. 6. Note that from the fourth step, the attack focuses two key windows instead of three because rotations lead to a position that has already been recovered. Finally, the last step considers the entire 32-bit output word using φ_3 and the known bits/collisions.

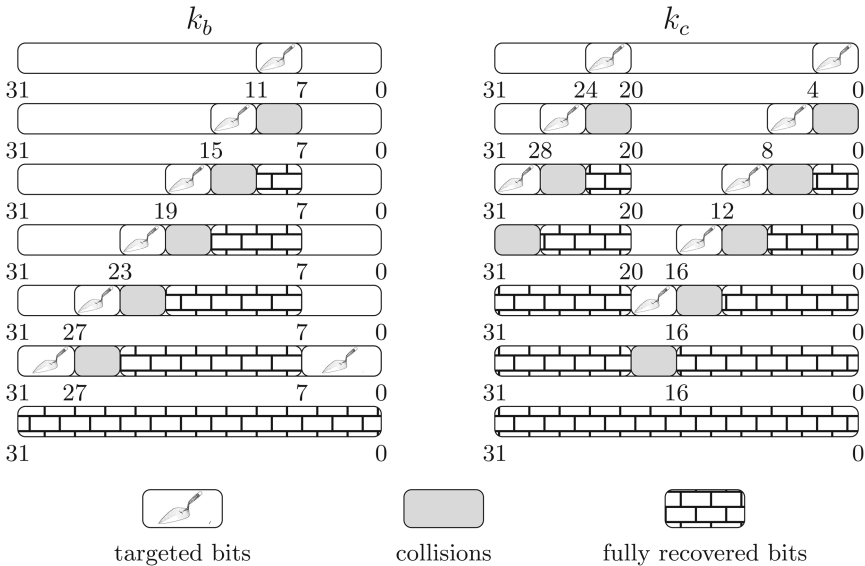


Fig. 6. Bricklayer attack example on IQR

6 Applications in Practice

6.1 Practical Experiments

All practical experiments presented below were done using an ARM 32-bit Cortex-M3 processor clocked at 24 MHz. Note that the DUT does not embed any hardware countermeasure against side-channel attacks. A trigger signal was inserted to indicate the beginning and the end of the penultimate round in order to avoid synchronization complications. EM emanations were measured using a Langer LF-U 5 near-field probe (100 kHz–50 MHz) and a LeCroy WaveSurfer 10 oscilloscope sampled at 10 GS/s. The signal was amplified using a Langer PA 303 BNC preamplifier, providing a gain of 30 dB. We used the same leakage model as for our simulations, since our microcontroller leaks the HW of intermediate values.

First, we tried to perform correlation analyses by focusing on arithmetic operations, without success. Figure 7 emphasizes that attacking the final block addition during executions of `quarterround'` was not successful, whereas for the compiled C version (which stores the intermediate values in RAM), we were able to retrieve the key bits. This reinforced our assumption that, depending on the computing platform, memory accesses can be the only source of exploitable leakage for software implementations.

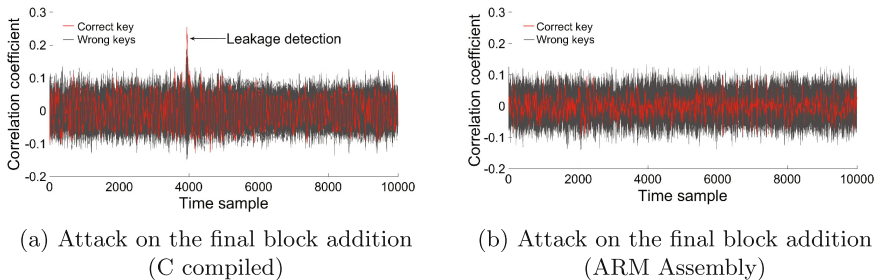


Fig. 7. Impact of memory accesses on electromagnetic leakage

In order to put the bricklayer attack into practice, the following hard-coded input block was used to encrypt 250 kB of data, where the counter (*i.e.* `nonce0`) was incremented for each 512-bit block (Fig. 8).

Figure 9 depicts all the correlation curves corresponding to each step of the bricklayer attack when targeting k_2 and k_7 . We incremented the windows' length by 4 at each step, exactly as illustrated in Fig. 6, resulting in an overall computational complexity of 2^{13} . All CEMAs were computed by halving the key search space. Consequently, some results do not appear clearly on charts and have to be deduced.

The first step, which targets $k_7^{23..20} \parallel k_7^{3..0} \parallel k_2^{10..7}$, returned the collisions $\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\} = \{56, 176, 2096, 2232\}$. For the next stages, each key

61707865	3320646e	79622d32	6b206574
ad0578e5	e962fc0a	42ffc031	75018bee
b7ae69dc	f1490ca8	89ac12fd	be8466d3
00000000	f1d69cbf	8e34191d	7024af3b

Fig. 8. Input block used for practical experiments

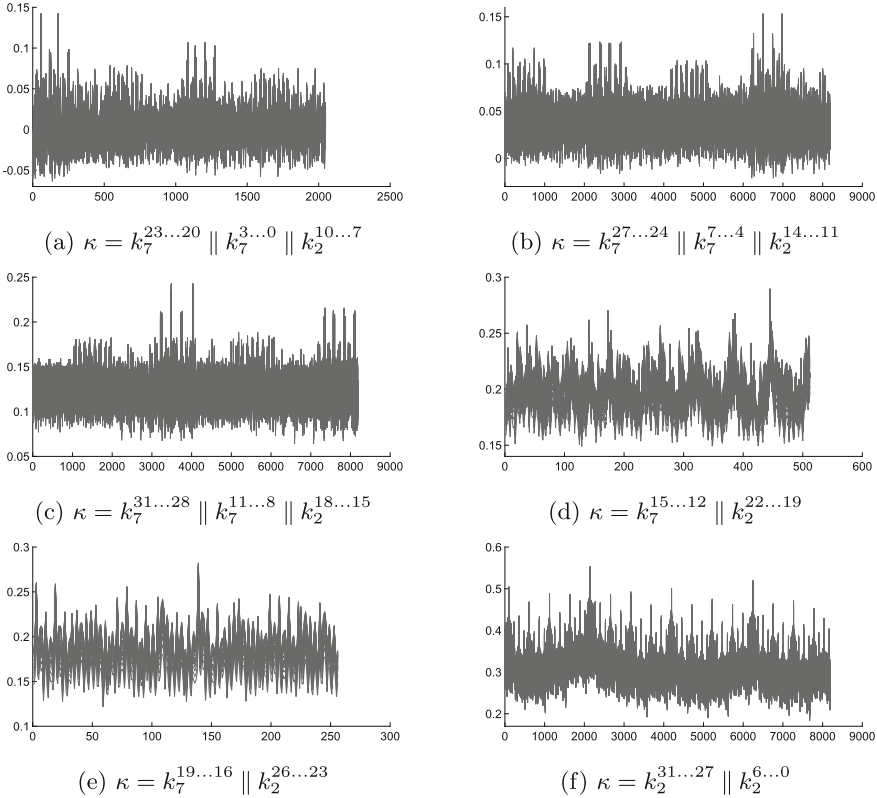


Fig. 9. CEMAs to recover k2 and k7

hypothesis $\kappa \in \mathcal{K}$ was coupled to each collision $\gamma_j \in \Gamma$ and was placed at the index $i = \kappa \cdot |\Gamma| + j$ of the prediction vector. Thus, higher coefficients at indexes i revealed the correct collision of the previous step γ_j by computing $j = i \bmod |\Gamma|$. Finally, the new collisions are equal to $(i - j) / |\Gamma|$. For instance, Fig. 9b indicates that the maximum coefficient appears at indexes $i \in \{6499, 6979\}$. Both indexes are congruent to 3 modulo 4, which means that $\gamma_3 = k_7^{23\dots20} \parallel k_7^{3\dots0} \parallel k_2^{10\dots7}$. As a result, the collisions for $k_7^{27\dots24} \parallel k_7^{7\dots4} \parallel k_2^{14\dots11}$ are defined by $\Gamma = \{1624, 1744, 3664, 3800\}$. The remaining steps followed the same methodology, making it possible to recover k_2 and k_7 entirely. Obviously, this can be applied on other IQRs in parallel to recover the whole encryption key.

A drawback of the D&C method is the number of required measurements, since the leakage of the omitted bits influences the attacked ones. Thus, more traces are needed in order to average out noise. Figure 10 compares, regarding the number of measurements, an attack on the QR using $\varphi_{2,3}$ with the first step of the bricklayer attack presented above, using the same measurement setup.

As a result, to recover the same number of key bits, $\varphi_{4,n}$ requires less traces as it targets larger windows than $\varphi_{2,n}$. However, the number of required traces decreases at each step of the bricklayer attack as the size of targeted windows increases.

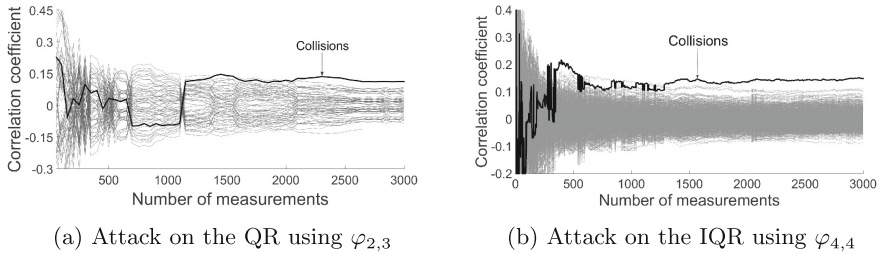


Fig. 10. Correlation coefficients to recover 12 key bits

6.2 Attacks' Feasibility on Existing Protocols

In a typical side-channel analysis, it is assumed that the attacker has access to either the plaintext or the ciphertext, but not necessarily to both. In the case of ChaCha, we can consider the knowledge of the nonce as the knowledge of the plaintext. However, attacks using $\varphi_{4,n}$ require the knowledge of the keystream (*i.e.* plaintexts and ciphertexts), in addition to nonces. This is a strong assumption that could be available in an evaluation laboratory but might be hard to set up in practice, leaving the attacks from nonces more realistic. Therefore, we discuss whether the knowledge of nonces is a fair assumption.

By definition, the single requirement for a cryptographic nonce is to be used only once. Therefore, a simple counter could suit the need. However, in cases where many different keys are used, some protocols (*e.g.* TLS) force a part of the nonce (*e.g.* the IV) to be random in order to thwart multi-key attacks [27]. This leaves the block counter as the only predictable part of the nonce. Therefore, if this latter is defined on n bits, then a correlation analysis cannot recover more than $2 \cdot n$ key bits. As a result, it introduces a protocol-level countermeasure which protects a large part of the key.

Still, existing protocols are not defined in this way. For instance, the Secure Shell (SSH) protocol uses the packet sequence number as a 64-bit IV [1] whereas the remaining 64 bits are used for the block counter, which is reset for each packet. Consequently, observing an entire SSH session makes it possible to predict the entire nonce, giving an attacker the opportunity to recover all key words as soon as enough packets are transmitted.

Furthermore, another construction that can be encountered in practice is XChaCha20, which is implemented in the Sodium crypto library [17]. This construction was first proposed for Salsa20 [6] and aims at extending the nonce to 192 bits so that it can be picked at random. The main idea is to encrypt a block with a fixed key k and 128 bits of the random nonce, without executing the final block addition. The first and last 16 bytes of the output result in a 256-bit subkey k' . Finally, the regular ChaCha20 algorithm is executed using the 64 remaining bits of the 192-bit nonce as IV, and k' as encryption key. Note that XChaCha20 is intrinsically resistant against attacks from the keystream, since the final block addition is omitted during the subkey generation. However, the 192-bit nonce must be transmitted in clear and can be entirely known by the attacker.

These real life case studies introduce the need of dedicated countermeasures against side-channel attacks when ChaCha is deployed in such conditions.

7 Towards a Secure Implementation

A common approach to thwart side-channel attacks is the use of *masking*. This countermeasure consists in blinding the processed values x by means of random masks r , so that intermediate variables are impossible to predict. Thus, an attacker has to analyze multiple point distributions, which exponentially increases the attack complexity with the number of shares. In this section, we only discuss first-order masking *i.e.* the case where a single mask is used to randomize the data. Because of their structures, ARX designs need both boolean ($x' = x \oplus r$) and arithmetic ($x' = x \boxplus r$) masking.

To overcome this complication, there are two main approaches. The first one is to switch from one masking scheme to the other whenever necessary. The first conversion algorithms, described by Goubin in [20], have complexity of $\mathcal{O}(1)$ for boolean to arithmetic and $\mathcal{O}(k)$ for arithmetic to boolean, where k refers to the addends' bit size. The latter was then improved by Coron *et al.* to $\mathcal{O}(\log k)$ [15]. The second approach is to directly perform an addition on the masked values, eliminating the need for conversions [22]. However, secure adders usually rely on the recursion formulae involved in arithmetic to boolean conversions. Consequently, they inherit from the same complexity.

The best method, in terms of performance, depends on the algorithm to be protected. For instance, masks conversions are more efficient when several arithmetic operations are processed successively, since only one arithmetic to boolean conversion is ultimately required. Otherwise, secure adders can lead to better performances as shown by a practical comparison between HMAC-SHA-1 and Speck in [15]. In order to give an insight into the overhead introduced by a first-order masking, we implemented two secure adders in C language, using the same compilation options as described in Sect. 4.3. This allowed us to compare, in terms of performance, our secure implementations of ChaCha20 with the one from the OpenSSL library. Running times given in Table 1 are expressed in clock cycles and were computed with the help of debug sessions. Note that these measurements do not take the generation of random numbers into account since this

operation depends a lot on the computing platform. As these countermeasures were implemented in C, they do not ensure the absence of memory accesses within QRs. On the other hand, handling all data in registers during a whole QR may not be possible, since masking also increases memory requirements. Further investigations need to be carried out to determine which algorithms could minimize memory access within QRs and how to securely manage them.

Table 1. Running time in clock cycles to encrypt a 512-bit block using ChaCha20 on an ARM Cortex-M3

	Time	Penalty factor
ChaCha20 unmasked	4 380	1
ChaCha20 with Karroumi <i>et al.</i> SecAdd [22]	121 618	28
ChaCha20 with Coron <i>et al.</i> SecAdd [15]	93 993	22

These practical results point out how difficult it is to effectively secure ARX ciphers’ implementations. However, masking is not the only answer to side-channel attacks and is often combined with *hiding* countermeasures. The principle of hiding is to randomize an algorithm execution by running its operations at different moments in time, during each execution [36, 39]. This can be achieved by randomly inserting dummy operations and *shuffling*. Shuffling intends to randomly change the sequence of operations that can be computed in arbitrary order. In practice, hiding countermeasures increase the number of traces needed to carry out an attack [16, 33].

Regarding ChaCha, operations within a QR cannot be shuffled as they are executed sequentially. On the other hand, each QR can be computed independently from the other, but this is only true for a single round because of switching from column to diagonal rounds. However, there are many ways to implement hiding in practice and further investigations will have to be carried out on the specific case of ChaCha.

8 Conclusions and Further Work

This paper presents side-channel analyses of ChaCha based on leakages related to memory accesses. Our study emphasizes that quantifying the signal available to the attacker at the instruction level could allow to strengthen implementations without much effort.

We compare, from a side-channel point of view, two different software implementations of ChaCha20 on a 32-bit processor. As a result, minimizing memory accesses makes selection functions more complex, to such an extent that they may lead to collisions. We introduce the bricklayer attack to defeat such implementations. Our results show that attacking the reverse QRs (*i.e.* from the keystream) is more efficient than attacking the regular ones (*i.e.* from the input

block). However, we highlight that attacks from the input block are the most pragmatic threats since the knowledge of the keystream is a strong assumption. Finally, we discuss possible countermeasures at several levels and highlight how expensive it is to implement first-order masking for ChaCha20 with practical measurements. Therefore, further work must be undertaken to propose efficient secure implementations of ChaCha.

References

1. chacha20-poly1305@openssh.com: Authenticated encryption mode, May 2016. <http://bvx.su/OpenBSD/usr/bin/ssh/PROTOCOL.chacha20poly1305>
2. iOS 10 Security White Paper. Technical report, Apple Inc., March 2017. https://www.apple.com/business/docs/iOS_Security_Guide.pdf
3. Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of Latin dances: analysis of Salsa, ChaCha, and Rumba. Cryptology ePrint Archive, Report 2007/472 (2007). <http://eprint.iacr.org/2007/472>
4. Babbage, S., Borghoff, J., Velichkov, V.: The eSTREAM portfolio in 2012. <http://www.ecrypt.eu.org/ecrypt2/documents/D.SYM.10-v1.pdf>
5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: SIMON and SPECK: block ciphers for the Internet of Things. Cryptology ePrint Archive, Report 2015/585 (2015). <http://eprint.iacr.org/2015/585>
6. Bernstein, D.J.: Extending the Salsa20 nonce. <https://cr.yp.to/snuffle/xsalsa-20081128.pdf>
7. Bernstein, D.J.: ChaCha, a variant of Salsa20. In: SASC - The State of the Art of Stream Ciphers, pp. 273–278 (2008). <http://cr.yp.to/chacha/chacha-20080128.pdf>
8. Bernstein, D.J.: The Salsa20 family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) *New Stream Cipher Designs*. LNCS, vol. 4986, pp. 84–97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_8
9. Biryukov, A., Dinu, D., Großschädl, J.: Correlation power analysis of lightweight block ciphers: from theory to practice. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) *ACNS 2016*. LNCS, vol. 9696, pp. 537–557. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_29
10. Boura, C., Lvque, S., Vigilant, D.: Side-channel analysis of Grostl and Skein. In: 2012 IEEE Symposium on Security and Privacy Workshops, pp. 16–26, May 2012. <https://www.ieee-security.org/TC/SPW2012/proceedings/4740a016.pdf>
11. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) *CHES 2004*. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
12. Bursztein, E.: Speeding up and strengthening HTTPS connections for Chrome on Android. Technical report, April 2014. <https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html>
13. Callan, R., Zajić, A., Prvulovic, M.: A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events. In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-47*, pp. 242–254. IEEE Computer Society, Washington, D.C. (2014). <http://dx.doi.org/10.1109/MICRO.2014.39>
14. Choudhuri, A.R., Maitra, S.: Differential cryptanalysis of Salsa and ChaCha - an evaluation with a hybrid model. Cryptology ePrint Archive, Report 2016/377 (2016). <http://eprint.iacr.org/2016/377>

15. Coron, J.-S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to boolean masking with logarithmic complexity. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 130–149. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_7
16. Couroussé, D., Barry, T., Robisson, B., Jaillon, P., Potin, O., Lanet, J.-L.: Runtime code polymorphism as a protection against side channel attacks. Cryptology ePrint Archive, Report 2017/699 (2017). <http://eprint.iacr.org/2017/699>
17. Denis, F.: The XChaCha20-Poly1305 construction. https://download.libsodium.org/doc/secret-key_cryptography/xchacha20-poly1305.construction.html
18. Dworkin, M.J.: SP 800-38A 2001 edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Technical report, Gaithersburg, MD, United States (2001)
19. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_21
20. Goubin, L.: A sound method for switching between boolean and arithmetic masking. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 3–15. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_2
21. Jungk, B., Bhasin, S.: Don't fall into a trap: physical side-channel analysis of chacha20-poly1305. In: Design, Automation Test in Europe Conference Exhibition (DATE 2017), pp. 1110–1115, March 2017
22. Karroumi, M., Richard, B., Joye, M.: Addition with blinded operands. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 41–55. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10175-0_4
23. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25. <http://dl.acm.org/citation.cfm?id=646764.703989>
24. Kumar, S.V.D., Patranabis, S., Breier, J., Mukhopadhyay, D., Bhasin, S., Chattopadhyay, A., Bakshi, A.: A practical fault attack on ARX-like ciphers with a case study on ChaCha20. In: 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTCT, Taipei, Taiwan (2017)
25. Langley, A., Chang, W., Mavrogiannopoulos, N., Strombergson, J., Josefsson, S.: ChaCha20-Poly1305 cipher suites for transport layer security (TLS). RFC 7905, RFC Editor, June 2016. <http://tools.ietf.org/rfc/rfc7905.txt>
26. Lemke, K., Schramm, K., Paar, C.: DPA on n -bit sized boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-construction. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 205–219. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_15. <http://www.iacr.org/archive/asiacrypt2007/31560191/31560191.pdf>
27. Luykx, A., Mennink, B., Paterson, K.G.: Analyzing multi-key security degradation. Cryptology ePrint Archive, Report 2017/435 (2017). <http://eprint.iacr.org/2017/435>
28. Maitra, S.: Chosen IV cryptanalysis on reduced round ChaCha and Salsa. Discrete Appl. Math. **208**(C), 88–97 (2016). <http://dx.doi.org/10.1016/j.dam.2016.02.020>
29. Mazumdar, B., Ali, S.S., Sinanoglu, O.: Power analysis attacks on ARX: an application to Salsa20. In: 2015 IEEE 21st International On-line Testing Symposium (IOLTS), pp. 40–43, July 2015

30. McCann, D., Eder, K., Oswald, E.: Characterising and comparing the energy consumption of side channel attack countermeasures and lightweight cryptography on embedded devices. *Cryptology ePrint Archive*, Report 2015/832 (2015). <http://eprint.iacr.org/2015/832>
31. McGrew, D., Bailey, D.: AES-CCM cipher suites for transport layer security (TLS). RFC 6655, RFC Editor, July 2012. <http://tools.ietf.org/rfc/rfc6655.txt>
32. Mozaffari-Kermani, M., Azarderakhsh, R.: Reliable hash trees for post-quantum stateless cryptographic hash-based signatures. In: 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp. 103–108, October 2015
33. Patranabis, S., Roy, D.B., Vadnala, P.K., Mukhopadhyay, D., Ghosh, S.: Shuffling across rounds: a lightweight strategy to counter side-channel attacks. In: 2016 IEEE 34th International Conference on Computer Design (ICCD), pp. 440–443, October 2016
34. Quisquater, J.-J., Samyde, D.: ElectroMagnetic Analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) *E-smart 2001*. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45418-7_17
35. Rescorla, E.: The transport layer security (TLS) protocol version 1.3. Internet-Draft draft-ietf-tls-tls13-21, Internet Engineering Task Force, July 2017. <https://tswg.github.io/tls13-spec/draft-ietf-tls-tls13.html>, work in Progress
36. Rivain, M., Prouff, E., Doget, J.: Higher-order masking and shuffling for software implementations of block ciphers. *Cryptology ePrint Archive*, Report 2009/420 (2009). <http://eprint.iacr.org/2009/420>
37. Salowey, J., Choudhury, A., McGrew, D.: AES Galois Counter Mode (GCM) cipher suites for TLS. RFC 5288, RFC Editor, August 2008. <http://www.rfc-editor.org/rfc/rfc5288.txt>
38. Shi, Z., Zhang, B., Feng, D., Wu, W.: Improved key recovery attacks on reduced-round Salsa20 and ChaCha. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) *ICISC 2012*. LNCS, vol. 7839, pp. 337–351. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37682-5_24
39. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.-X.: Shuffling against side-channel attacks: a comprehensive study with cautionary note. In: Wang, X., Sako, K. (eds.) *ASIACRYPT 2012*. LNCS, vol. 7658, pp. 740–757. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_44
40. Yadav, P., Gupta, I., Murthy, S.K.: Study and analysis of eSTREAM cipher Salsa and ChaCha. In: 2016 IEEE International Conference on Engineering and Technology (ICETECH), pp. 90–94, March 2016
41. Zohner, M., Kasper, M., Stöttinger, M.: Butterfly-attack on Skein’s modular addition. In: Schindler, W., Huss, S.A. (eds.) *COSADE 2012*. LNCS, vol. 7275, pp. 215–230. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29912-4_16

CCA-secure Predicate Encryption from Pair Encoding in Prime Order Groups: Generic and Efficient

Sanjit Chatterjee, Sayantan Mukherjee^(✉), and Tapas Pandit

Department of Computer Science and Automation, Indian Institute of Science,
Bangalore, India

{sanjit,sayantamm,tapas}@iisc.ac.in

Abstract. Attrapadung (Eurocrypt 2014) proposed a generic framework called pair encoding to simplify the design and proof of security of CPA-secure predicate encryption (PE) in composite order groups. Later Attrapadung (Asiacrypt 2016) extended this idea in prime order groups. Yamada et al. (PKC 2011, PKC 2012) and Nandi et al. (ePrint Archive: 2015/457, AAECC 2017) proposed generic conversion frameworks to achieve CCA-secure PE from CPA-secure PE provided the encryption schemes have properties like delegation or verifiability. The delegation property is harder to achieve and verifiability based conversion degrades the decryption performance due to a large number of additional pairing evaluations. Blömer et al. (CT-RSA 2016) proposed a direct fully CCA-secure predicate encryption in composite order groups but it was less efficient as it needed a large number of pairing evaluations to check ciphertext consistency. As an alternative, Nandi et al. (ePrint Archive: 2015/955) proposed a direct conversion technique in composite order groups. We extend the direct conversion technique of Nandi et al. in the prime order groups on the CPA-secure PE construction by Attrapadung (Asiacrypt 2016) and prove our scheme to be CCA-secure in a quite different manner. Our first direct CCA-secure predicate encryption scheme requires exactly one additional ciphertext component and three additional units of pairing evaluation during decryption. The second construction requires exactly three additional ciphertext components but needs only one additional unit pairing evaluation during decryption. This is a significant improvement over conventional approach for CPA-to-CCA conversion in prime order groups.

1 Introduction

Predicate encryption (PE) is a new paradigm for public-key encryption that evaluates a predicate function $R : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$ in the encrypted domain. Informally, in a PE system, a ciphertext \mathbf{C} is associated with a data-index $y \in \mathcal{Y}$, a secret key \mathbf{K} is associated with a key-index $x \in \mathcal{X}$ and the secret key \mathbf{K} can decrypt the ciphertext \mathbf{C} if and only if $R(x, y) = 1$. The simplest example is IBE [1] where R is an equality predicate.

Waters [2] introduced the dual system technique to construct adaptively secure predicate encryption schemes. Attrapadung [3] and Wee [4] independently observed a similarity in the structure of the proofs of dual system technique based adaptively secure predicate encryption schemes. The notions of *pair encoding* [3] and *predicate encoding* [4] were introduced as abstraction of complex key and ciphertext structure of available predicate encryptions. Such encodings allowed them to construct adaptively CPA-secure predicate encryptions using dual system technique. This new approach not only allowed them to improve the performance of several available predicate encryption schemes but also to instantiate several completely new schemes. For example, pair encoding allowed the first-ever construction of PE for regular language, ABE with constant-size ciphertext etc. as presented in [3]. However, all these CPA-secure predicate encryptions were constructed in composite order groups.

Later Attrapadung [5] and Chen et al. [6] constructed adaptive CPA-secure predicate encryption schemes in the prime order groups using pair encoding and predicate encoding respectively. The construction [6] was even more modular due to the use of *dual system group* (DSG) [7]. Agrawal et al. [8,9] integrated pair encoding and dual system group and introduced different security notions for pair encoding.

Motivation. All the aforementioned schemes aim at constructing CPA-secure predicate encryption. In various practical scenarios, however, CCA-security is assumed to be mandatory. One can use available generic techniques [10–13] to convert CPA-secure predicate encryption into CCA-secure predicate encryption. Informally these techniques add new components in the CPA-ciphertext that can be used later to check if the ciphertext has been tampered in the line. They therefore face problems of two-fold – (1) increased length of key-indices and data-indices which result in a *bigger* secret key and ciphertext due to *index transformation* [10,12] and (2) extra cost to perform verifiability or delegation. We consider verifiability based approach as a benchmark since delegation is not known for most of the predicate encryptions. For example, verifiability based solution makes the decryption lot costlier than the cost of decryption in the CPA-secure scheme in terms of the number of pairings evaluated. Blömer et al. [14] proposed a direct CCA-secure predicate encryption from pair encodings in composite order groups. Their verifiability based check requires additional pairing operations which is nearly same as that required in underlying CPA-decryption. As an alternative, Nandi et al. [15] suggested a direct conversion to CCA-secure predicate encryptions from pair encodings. Even though that conversion is efficient and generic, it works in the composite order group. Naturally one would like to construct a direct CCA-secure predicate encryption in prime order groups from a CPA-secure predicate encryption without compromising the performance. [16] and its descendants suggested certain frameworks to convert pairing based cryptosystems from composite order to prime order. However, to the best of our knowledge, such frameworks are not directly applicable for CPA to CCA conversion.

Our Contribution. In this work, we consider the pair encoding based CPA-secure predicate encryptions construction of [3,5]. We propose two generic constructions

of direct CCA-secure PE from pair encoding based CPA-secure PE. Both of our constructions are achieved in prime order group and neither uses the trick called *index transformation* [10, 12]. This results in more efficient CCA-secure PE in terms of the size of the ciphertext and number of pairing evaluations during decryption. Roughly speaking, given a CPA-secure predicate encryption, we create a hash of CPA-ciphertext and extend the idea of *injective encoding* [17] that adds a new ciphertext component as a “commitment” of CPA-ciphertext. We call this construction as *direct* CCA-secure construction as we do not conform to the traditional two-step approach of *index transformation* followed by *delegation* or *verifiability*.

Our first construction adds *only one* additional component to the ciphertext of [5] at the cost of *three* additional unit pairing evaluations during decryption. The second construction however is more efficient in terms of the number of pairing evaluations during decryption. The ciphertext in this construction adds *exactly three* additional components to the ciphertext of [5] namely a $(d+1)$ -tuple made up of source group elements (i.e. an element of $\mathbb{G} = G_1^{(d+1)}$), an one-time signature (OTS) verification key and a signature. During decryption this construction needs *only one* additional unit of pairing evaluation along with an OTS verification. As we can see in Table 1, our generic techniques to construct CCA-secure predicate encryptions enjoy smaller (constant) increase in ciphertext size that naturally results in less number of pairing evaluations during decryption.

Table 1. Comparative study of efficiency (For delegation/verifiability-based conversions, $x' \leftarrow \mathcal{T}_1(x)$, $x_{vk} \leftarrow \mathcal{T}_2(x, vk)$, $y_{vk} \leftarrow \mathcal{T}_3(y, vk)$ are transformed indices [13] and usually quite larger than input indices (x or y). Each ciphertext or key-component is a $(d+1)$ -dimensional vector from $\mathbb{G} = G_1^{(d+1)}$ and $\mathbb{H} = G_2^{(d+1)}$ respectively. \mathcal{C} denotes the cost function. The two-columns under “Decryption Cost” follows the convention that the *first* cell is underlying CPA-Decryption cost and *second* cell contains *additional* cost to achieve CCA-Decryption.).

Technique	Key	Ciphertext	Decryption cost	
<i>Delegation-based</i>	$\mathcal{O}(x')$	$\mathcal{O}(y_{vk})$	$\mathcal{C}(\text{Decrypt}(\mathbf{K}_{x_{vk}}, \mathbf{C}_{y_{vk}}))$	$\mathcal{C}(\text{Delegate}(\mathbf{K}_{x'}, x', x_{vk}))$
<i>Verifiability-based</i>			$\mathcal{C}(\text{Decrypt}(\mathbf{K}_{x'}, \mathbf{C}_{y_{vk}}))$	$\mathcal{C}(\text{Verify}(\mathbf{C}_{y_{vk}}, x', \epsilon_{vk}))$
Π_R (Sect. 3.2)	$\mathcal{O}(x)$	$\mathcal{O}(y)$	$\mathcal{C}(\text{Decrypt}(\mathbf{K}_x, \mathbf{C}_y))$	3 unit pairing
Π'_R (Sect. 3.4)	$\mathcal{O}(x)$	$\mathcal{O}(y)$	$\mathcal{C}(\text{Decrypt}(\mathbf{K}_x, \mathbf{C}_y))$	1 unit pairing

The idea of using OTS (resp. injective encoding) to achieve CCA-secure PKE/(H)-IBE was first introduced in [18] (resp. [17]). Our approach, while bearing some similarity, differ significantly from [17, 18] and their follow-up works in the context of Id-based encryption. This is because, the structure of pair encoding based general PE is much more involved than that of simpler equality predicate in IBE. The primary achievement of this paper over [15] is amalgamation of [17] (resp. [18]) with prime-order matrix-based construction of predicate encryption [5]. Our techniques (Lemmas 1 and 2) demonstrate that simple primitives like [17, 18] can still be deployed to achieve CCA-security even when the ciphertext/key is of complicated matrix structure.

Organization of the Paper. Section 2 contains necessary definitions and notations that are followed in this paper. In Sect. 3 we describe two constructions to achieve CCA-secure predicate encryption. Section 4 concludes the paper.

2 Preliminaries

Notations. We denote $[a, b] = \{i \in \mathbb{N} : a \leq i \leq b\}$ and $[n] = [1, n]$. We assume \mathbf{v} is a vector having components v_1, \dots, v_n . By $s \stackrel{\$}{\leftarrow} S$ we denote a uniformly random choice s from set S . 1^λ denotes the security parameter for $\lambda \in \mathbb{N}$. Any $\mathbf{x} \in S^{\mathcal{k}}$ is a \mathcal{k} -dimensional column vector. We use both $\mathbf{x} \in S^{1 \times \mathcal{k}}$ and $\mathbf{x} \in (S)^{\mathcal{k}}$ to denote \mathcal{k} -dimensional row vectors. $\mathbb{GL}_{N, \ell}$ is the group of non-singular matrices with dimension $\ell \times \ell$ and the scalars are from \mathbb{Z}_N .

Predicate Family. The predicate family for an index family \mathcal{X} is $\mathcal{R} = \{R_\kappa\}_{\kappa \in \mathcal{X}}$, where $R_\kappa : \mathcal{X}_\kappa \times \mathcal{Y}_\kappa \rightarrow \{0, 1\}$ is a predicate function and \mathcal{X}_κ and \mathcal{Y}_κ are key-space and data-space respectively. We will often omit κ in the subscript for the simplicity of representation.

2.1 Predicate Encryption

A predicate encryption (PE) scheme Π_R for predicate function $R : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$ consists of following algorithms.

- **Setup**($1^\lambda, \kappa$) for the security parameter $\lambda \in \mathbb{N}$ generates master secret key msk and public key mpk .
- **KeyGen**(msk, x) generates secret key \mathbf{K} of the given key-index $x \in \mathcal{X}$.
- **Encrypt**(mpk, y, M) takes as input data-index $y \in \mathcal{Y}$ and a message $M \in \mathcal{M}$ and generates ciphertext \mathbf{C} .
- **Decrypt**(\mathbf{K}, \mathbf{C}) takes a key \mathbf{K} corresponding to key-index x and a ciphertext \mathbf{C} corresponding to data-index y and outputs a message M or \perp .

Correctness. A predicate encryption scheme is said to be correct if for all $(mpk, msk) \leftarrow \text{Setup}(1^\lambda, \kappa)$, all $y \in \mathcal{Y}$, all $M \in \mathcal{M}$, all $\mathbf{C} \leftarrow \text{Encrypt}(mpk, y, M)$, all $x \in \mathcal{X}$, all $\mathbf{K} \leftarrow \text{KeyGen}(msk, x)$,

$$\text{Decrypt}(\mathbf{K}, \mathbf{C}) = \begin{cases} M & \text{if } R(x, y) = 1 \\ \perp & \text{if } R(x, y) = 0. \end{cases}$$

Security. Chosen ciphertext security (IND-CCA) of a predicate encryption scheme Π_R can be modeled as a security game between challenger \mathcal{C} and adversary \mathcal{A} .

- **Setup:** \mathcal{C} gives out mpk and keeps msk as secret.
- **Phase-I Query:** Queries are performed to available oracles as follows.

- **Key Query:** Keygen oracle \mathcal{O}_K returns $\mathbf{K} \leftarrow \text{KeyGen}(msk, x)$ for a given key-index x .
- **Dec Query:** Given (x, \mathbf{C}) , decryption oracle \mathcal{O}_D returns $\text{Decrypt}(\mathbf{K}, \mathbf{C})$.
- **Challenge:** \mathcal{A} provides challenge data-index y^* (such that $R(x, y^*) = 0$ for all key query x) and two messages (M_0, M_1) of equal length. \mathcal{C} generates $\mathbf{C}^* \leftarrow \text{Encrypt}(mpk, y^*, M_b)$ for $b \xleftarrow{\$} \{0, 1\}$.
- **Phase-II Query:** Queries are performed to available oracles as follows.
 - **Key Query:** Given a key-index x such that $R(x, y^*) = 0$, keygen oracle \mathcal{O}_K returns $\mathbf{K} \leftarrow \text{KeyGen}(msk, x)$.
 - **Dec Query:** Given (x, \mathbf{C}) , decryption oracle \mathcal{O}_D returns $\text{Decrypt}(\mathbf{K}, \mathbf{C})$ if the conditions $R(x, y^*) = 1$ and $\mathbf{C} = \mathbf{C}^*$ are not satisfied together.
- **Guess:** \mathcal{A} outputs its guess $b' \in \{0, 1\}$ and wins if $b = b'$.

For any adversary \mathcal{A} the advantage is,

$$\text{Adv}_{\mathcal{A}}^{IR}(\lambda) = |\Pr[b = b'] - 1/2|.$$

A predicate encryption scheme is said to be IND-CCA secure if for any efficient adversary \mathcal{A} , $\text{Adv}_{\mathcal{A}}^{IR}(\lambda) \leq \text{neg}(\lambda)$. If the decryption oracle is not available to the adversary, we call such security model as IND-CPA security model.

2.2 Pair Encoding Schemes

Attrapadung [3] introduced the notion of pair encoding scheme which was later [5] refined with the properties called *regularity* of pair encoding. Here we recall the definition of pair encoding [3] and will discuss regular properties of pair encoding in Sect. 3.1.

A Pair Encoding P for a predicate function $R_\kappa : \mathcal{X}_\kappa \times \mathcal{Y}_\kappa \rightarrow \{0, 1\}$ indexed by $\kappa = (N \in \mathbb{N}, \text{par})$ consists of four deterministic algorithms:

- $\text{Param}(\kappa) \rightarrow n$ which is number of *common variables* $\mathbf{w} = (w_1, \dots, w_n)$ in EncK and EncC .
- $\text{EncK}(x, N) \rightarrow (\mathbf{k} = (k_1, \dots, k_{m_1}); m_2)$ where each k_ι for $\iota \in [m_1]$ is a polynomial of m_2 local variables $\mathbf{r} = (r_1, \dots, r_{m_2})$, common variables \mathbf{w} and private variable α .

$$k_\iota(\alpha, \mathbf{r}, \mathbf{w}) = b_\iota \alpha + \sum_{j \in [m_2]} b_{\iota j} r_j + \sum_{\substack{j \in [m_2] \\ k \in [n]}} b_{\iota j k} r_j w_k$$

- where $b_\iota, b_{\iota j}, b_{\iota j k} \in \mathbb{Z}_N$ for all $\iota \in [m_1]$, all $j \in [m_2]$ and all $k \in [n]$.
- $\text{EncC}(y, N) \rightarrow (\mathbf{c} = (c_1, \dots, c_{w_1}); w_2)$ where each $c_{\tilde{\iota}}$ for $\tilde{\iota} \in [w_1]$ is a polynomial of $(w_2 + 1)$ local variables $\mathbf{s} = (s_0, \dots, s_{w_2})$ and common variables \mathbf{w} .

$$c_{\tilde{\iota}}(\mathbf{s}, \mathbf{w}) = \sum_{j \in [0, w_2]} a_{\tilde{\iota} j} s_j + \sum_{\substack{j \in [0, w_2] \\ k \in [n]}} a_{\tilde{\iota} j k} s_j w_k$$

where $a_{\tilde{\iota} j}, a_{\tilde{\iota} j k} \in \mathbb{Z}_N$ for all $\tilde{\iota} \in [w_1]$, all $j \in [0, w_2]$ and all $k \in [n]$.

– $\text{Pair}(x, y, N) \rightarrow \mathbf{E} \in \mathbb{Z}_N^{m_1 \times w_1}$.

Correctness. A pair encoding scheme is said to be correct if for all $N \in \mathbb{N}$, for all $y \in \mathcal{Y}_\kappa$, $\mathbf{c} \leftarrow \text{EncC}(y, N)$, all $x \in \mathcal{X}_\kappa$, $\mathbf{k} \leftarrow \text{EncK}(x, N)$ and $\mathbf{E} \leftarrow \text{Pair}(x, y, N)$, $\mathbf{kEc}^\top = \alpha s_0$ if $R(x, y) = 1$.

Security. The pair encoding schemes [3, 5] achieve several security properties like *perfect master-key hiding* (PMH-Security), *co-selective master-key hiding* (CMH-Security) and *selective master-key hiding* (SMH-Security). For definition, see [3, 5] or the full version of this paper [19].

3 CCA-secure Predicate Encryption from Pair Encoding

Here we present two *direct* constructions of CCA-secure predicate encryption scheme in prime-order groups from pair encoding scheme.

3.1 Regular Decryption Pair Encoding

As recalled earlier, the notion of *regular* properties of pair encoding was introduced in [5, Definition 1]. We also note that the decryption sufficiency property was discussed in [15, Conditions 3.1]. Here, we need the pair encoding to satisfy both these properties. We call a pair encoding that satisfies all the above mentioned properties, *regular decryption* pair encoding. The regular decryption properties of a pair encoding are listed below. The first four (precisely Properties $\mathcal{P}1, \mathcal{P}2, \mathcal{P}3, \mathcal{P}4$) denote the regular properties of pair encoding. Note that, these restrictions are quite natural and are observed in all the available pair encoding based predicate encryption constructions [3–6, 8, 9]. The regular decryption properties of pair encoding are noted below:

- ($\mathcal{P}1$) : For $\tilde{\iota} \in [w_1], \iota \in [m_1]$, if $\exists j' \in [0, w_2], k' \in [n], j \in [m_2], k \in [n]$ such that $a_{\tilde{\iota}j'k'} \neq 0$ and $b_{\iota jk} \neq 0$, then $\mathbf{E}_{\tilde{\iota}\iota} = 0$.
- ($\mathcal{P}2$) : For $\iota \in [m_1]$, if $\exists j \in [m_2], k \in [n]$ such that $b_{\iota jk} \neq 0$ then $\exists \hat{\iota} \in [m_1]$ such that $k_{\hat{\iota}} = r_j$.
- ($\mathcal{P}3$) : For $\tilde{\iota} \in [w_1]$, if $\exists j' \in [0, w_2], k' \in [n]$ such that $a_{\tilde{\iota}j'k'} \neq 0$ then $\exists \hat{\iota} \in [w_1]$ such that $c_{\hat{\iota}} = s_j$.
- ($\mathcal{P}4$) : $c_1(\mathbf{s}, \mathbf{w}) = s_0$.
- ($\mathcal{P}5$) : For $(x, y) \in \mathcal{X} \times \mathcal{Y}$, such that $R(x, y) = 1$, $(\mathbf{k}; m_2) \leftarrow \text{EncK}(x, N)$ and $\mathbf{E} \leftarrow \text{Pair}(x, y, N)$ then $\mathbf{k}(\alpha, \mathbf{0}, \mathbf{0})\mathbf{E} = (*, 0, \dots, 0) \in \mathbb{Z}_N^{w_1}$ where $*$ is any non-zero entry.

Here we give some intuitive idea of regular decryption property of pair encoding. In Attrapadung’s prime-order instantiation of pair encoding based predicate encryption schemes, a particular type of commutativity was impossible to compute [5, Eq. (8)]. We use Property $\mathcal{P}1$ to restrict such cases. This property has been used to prove the correctness of the scheme. Property $\mathcal{P}2$ and $\mathcal{P}3$ ensure that if the key-encoding \mathbf{k} (resp. \mathbf{c}) contains $h_k r_j$ (resp. $h_{k'} s_{j'}$) then r_j (resp.

$s_{j'}$) has to be given explicitly. These two properties have been used in the security proof. We will see in the coming section that we produce a commitment on the *CPA-ciphertext* and bind it to the randomness s_0 . Therefore we fix the position of polynomial s_0 in Property $\mathcal{P}4$. Also to decrypt, given a secret key $\mathbf{K} \in (G_2^{(d+1)})^{m_1}$ and a pairing matrix $\mathbf{E} \in \mathbb{Z}_N^{m_1 \times w_1}$ (see Sect. 2.2 for description), the decryptor will compute an altKey $\hat{\mathbf{K}} = (\hat{\mathbf{K}}_0, \hat{\mathbf{K}}_1, \dots, \hat{\mathbf{K}}_{w_1}) \in (G_2^{(d+1)})^{(w_1+1)}$. We restrict that α used in secret key (\mathbf{K}) generation affects only $\hat{\mathbf{K}}_1$ via Property $\mathcal{P}5$. We will be needing this property in the security argument.

Next we describe our first construction and prove its security. Later we describe another construction that achieves better efficiency during decryption. Both of the constructions are developed on top of [5] as described in Remark 2.

3.2 Construction Π_R : Smaller Ciphertext

Given a pair encoding scheme P for predicate function R , a predicate encryption Π_R for the same predicate function R is defined as following.

- **Setup**($1^\lambda, N$): Runs $(G_1, G_2, G_T, e, p) \leftarrow \mathcal{G}(\lambda)$ where \mathcal{G} is an asymmetric prime-order bilinear group generator. Picks $(g_1, g_2) \xleftarrow{\$} G_1 \times G_2$. Runs $n \leftarrow \text{Param}(\kappa)$. Defines $\mathbb{W} = (\mathbf{W}_1, \dots, \mathbf{W}_{n+2})$ where $\mathbf{W}_i \xleftarrow{\$} \mathbb{Z}_p^{(d+1) \times (d+1)}$ for each $i \in [n+2]$. Chooses $(\mathbf{B}, \tilde{\mathbf{D}}, \alpha) \xleftarrow{\$} \mathbb{GL}_{p, d+1} \times \mathbb{GL}_{p, d} \times \mathbb{Z}_p^{(d+1)}$. Defines $\mathbf{D} = \begin{pmatrix} \tilde{\mathbf{D}} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$, $\mathbf{Z} = \mathbf{B}^{-\top} \mathbf{D}$, chooses collision resistant hash function $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbb{Z}_p$. Keeps $msk = (g_2^\alpha, \mathbf{B}, \mathbf{Z}, \mathbb{W})$ to be secret and computes,

$$mpk = \left(g_1^{\mathbf{B} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}, g_1^{\mathbf{W}_1 \mathbf{B} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}, \dots, g_1^{\mathbf{W}_{n+2} \mathbf{B} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}, g_2^{\mathbf{Z} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}, g_2^{\mathbf{W}_1^\top \mathbf{Z} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}, \dots, g_2^{\mathbf{W}_{n+2}^\top \mathbf{Z} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}, e(g_1, g_2)^{\alpha^\top \mathbf{B} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}, \mathcal{H} \right).$$

- **KeyGen**(msk, x): Runs $(\mathbf{k} = (k_1, \dots, k_{m_1}); m_2) \leftarrow \text{EncK}(x, N)$. Chooses $\mathbf{r}_1, \dots, \mathbf{r}_{m_2} \xleftarrow{\$} \mathbb{Z}_p^d$ and defines $\mathbf{R} = \left(\begin{pmatrix} \mathbf{r}_1 \\ \mathbf{0} \end{pmatrix}, \dots, \begin{pmatrix} \mathbf{r}_{m_2} \\ \mathbf{0} \end{pmatrix} \right) \in \left(\mathbb{Z}_p^{(d+1)} \right)^{m_2}$. Outputs $\mathbf{K} = \{g_2^{k_\iota(\alpha, \mathbf{R}, \mathbb{W})}\}_{\iota \in [m_1]} \in (G_2^{(d+1)})^{m_1}$ where for each $\iota \in [m_1]$,

$$k_\iota(\alpha, \mathbf{R}, \mathbb{W}) = b_\iota \alpha + \sum_{j \in [m_2]} b_{\iota j} \mathbf{Z} \begin{pmatrix} \mathbf{r}_j \\ \mathbf{0} \end{pmatrix} + \sum_{\substack{j \in [m_2] \\ k \in [n]}} b_{\iota j k} \mathbf{W}_k^\top \mathbf{Z} \begin{pmatrix} \mathbf{r}_j \\ \mathbf{0} \end{pmatrix}.$$

- **Encrypt**(mpk, y, M): Runs $(\mathbf{c} = (c_1, \dots, c_{w_1}); w_2) \leftarrow \text{EncC}(y, N)$. Chooses $\mathbf{s}_0, \dots, \mathbf{s}_{w_2} \xleftarrow{\$} \mathbb{Z}_p^d$ and defines $\mathbf{S} = \left(\begin{pmatrix} \mathbf{s}_0 \\ \mathbf{0} \end{pmatrix}, \dots, \begin{pmatrix} \mathbf{s}_{w_2} \\ \mathbf{0} \end{pmatrix} \right) \in \left(\mathbb{Z}_p^{(d+1)} \right)^{(w_2+1)}$. Computes $\mathbf{C} = (C_1, \dots, C_{w_1}, C_{w_1+1})$ where for each $\tilde{\iota} \in [w_1]$, $C_{\tilde{\iota}} = g_1^{c_{\tilde{\iota}}(\mathbf{S}, \mathbb{W})} \in G_1^{(d+1)}$ such that

$$c_{\tilde{l}}(\mathbf{S}, \mathbb{W}) = \sum_{j \in [0, w_2]} a_{\tilde{l}j} \mathbf{B} \begin{pmatrix} s_j \\ 0 \end{pmatrix} + \sum_{\substack{j \in [0, w_2] \\ k \in [n]}} a_{\tilde{l}jk} \mathbf{W}_k \mathbf{B} \begin{pmatrix} s_j \\ 0 \end{pmatrix} \text{ for } \tilde{l} \in [w_1]$$

and $C_{w_1+1} = M \cdot e(g_1, g_2)^{\alpha^\top \mathbf{B} \begin{pmatrix} s_0 \\ 0 \end{pmatrix}}$. It outputs $\overline{\mathbf{C}} = (\overline{\mathbf{C}}_0, \mathbf{C})$ where $\xi = \mathcal{H}(\mathbf{C})$ and $\overline{\mathbf{C}}_0 = g_1^{(\xi \mathbf{W}_{n+1} + \mathbf{W}_{n+2}) \mathbf{B} \begin{pmatrix} s_0 \\ 0 \end{pmatrix}}$.

- **Decrypt**($\mathbf{K}, \overline{\mathbf{C}}$): Given \mathbf{K} and $\overline{\mathbf{C}}$ corresponding to key-index x and data-index y respectively, if $R(x, y) = 0$, it aborts. It then computes $\xi = \mathcal{H}(\mathbf{C})$. It aborts if Eq. (1) is not satisfied.

$$e(\overline{\mathbf{C}}_0, g_2^{\mathbf{z} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}) = e(C_1, g_2^{(\xi \mathbf{W}_{n+1}^\top + \mathbf{W}_{n+2}^\top) \mathbf{z} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}). \quad (1)$$

Then runs $\mathbf{E} \leftarrow \text{Pair}(x, y, N)$. Given $\mathbf{K} = (K_1, \dots, K_{m_1})$ and ciphertext $\overline{\mathbf{C}}$ it computes $(\tilde{K}_1, \dots, \tilde{K}_{w_1})$ where $\tilde{K}_{\tilde{l}} = \prod_{\iota \in [m_1]} (K_\iota)^{\mathbf{E}_{\iota \tilde{l}}}$ for each $\tilde{l} \in [w_1]$. Chooses

$\mathbf{r} \xleftarrow{\$} \mathbb{Z}_p^d$. Defines modified key $\hat{\mathbf{K}} = (\hat{K}_0, \hat{K}_1, \dots, \hat{K}_{w_1})$ where $\hat{K}_0 = g_2^{\mathbf{z} \begin{pmatrix} \mathbf{r} \\ \mathbf{0} \end{pmatrix}}$, $\hat{K}_1 = \Phi \cdot \tilde{K}_1$ for $\Phi = g_2^{(\xi \mathbf{W}_{n+1}^\top + \mathbf{W}_{n+2}^\top) \mathbf{z} \begin{pmatrix} \mathbf{r} \\ \mathbf{0} \end{pmatrix}}$ and $\xi = \mathcal{H}(\mathbf{C})$ and $\hat{K}_i = \tilde{K}_i$ for $i \in [2, w_1]$. Outputs M such that

$$M = C_{w_1+1} \cdot e(\overline{\mathbf{C}}_0, \hat{K}_0) \cdot \left(\prod_{\tilde{l} \in [w_1]} e(C_{\tilde{l}}, \hat{K}_{\tilde{l}}) \right)^{-1}. \quad (2)$$

Correctness. See full version of this paper [19].

Remark 1. **Decrypt** creates *modified key* $\hat{\mathbf{K}}$ for a given secret key \mathbf{K} and the pairing matrix \mathbf{E} . From now onwards, we will use *decryption key* or *altKey* interchangeably to denote the *modified key*. We define $\text{AltKeyGen}(\overline{\mathbf{C}}, x, msk)$ to compute modified key $\hat{\mathbf{K}}$ and $\text{AltDecrypt}(\overline{\mathbf{C}}, \hat{\mathbf{K}})$ computes RHS of Eq. (2). This essentially divides the functionality of **Decrypt** function as composition of these two functions and helps us to process decryption queries during the proof.

Remark 2. We have extended the CPA-secure predicate encryption construction of [5]. The common variables \mathbb{W} , in our construction, contain $n+2$ matrices whereas in [5] it contained n matrices. These extra two common variables \mathbf{W}_{n+1} and \mathbf{W}_{n+2} are used to compute a commitment of CPA-ciphertext \mathbf{C} . A hash of \mathbf{C} is computed first and is binded to the randomness $\mathbf{B} \begin{pmatrix} s_0 \\ 0 \end{pmatrix}$ using common variables \mathbf{W}_{n+1} and \mathbf{W}_{n+2} . This results in an extra ciphertext component, namely, $\overline{\mathbf{C}}_0$. We then output $\overline{\mathbf{C}} = (\overline{\mathbf{C}}_0, \mathbf{C})$ as ciphertext. Notice that **KeyGen** algorithm is exactly the same as [5]. The **Decrypt** is modified to perform cancellation of the component $\overline{\mathbf{C}}_0$. To do that, we define *altKey* $\hat{\mathbf{K}}$ to contain \hat{K}_0 and Φ . We

use *associativity* [5, Sect. 4.1] to cancel the extra ciphertext component \overline{C}_0 using \hat{K}_0 and Φ . The cancellation is performed by introducing an extra unit of pairing evaluation $e(\overline{C}_0, \hat{K}_0)$ during decryption. Once such cancellation is performed, the decryption happens exactly like [5].

Efficiency. We introduce an extra check in Eq. (1) to ensure \overline{C}_0 to have a particular structure. The check in Eq. (1) incurs additional $2 \times (d+1)$ pairing evaluations. Therefore our construction incurs $3 \times (d+1)$ pairing evaluations during decryption in addition to pairing evaluation involved in CPA-ciphertext decryption [5]. This is really efficient as opposed to traditional CPA to CCA conversions by [10, 11, 13] that need roughly two times $m_1 \times w_1 \times (d+1) \times (m_2+1) \times d$ many pairing evaluations. We have discussed exact cost of such conversions in the full version [19].

Remark 3. Our construction uses a structure similar to the injective encoding first introduced in [7] to achieve CCA-secure PKE/(H)IBE from CPA-secure (H)IBE. However, the application of such a structure is far from straight forward as the ciphertext consistency check in Eq. (1) above may result in false-positives due to the complicated matrix-based structures in the ciphertext. We deal with this issue in Lemma 1.

3.3 Security of Π_R

To prove our predicate encryption construction (Π_R) fully CCA-secure, we extend the proof technique of Attrapadung [5]. In dual system proof technique, one needs to add randomness to ciphertext, keys and altKeys to construct semi-functional ciphertext, semi-functional key and semi-functional altKeys respectively. At the end, one has to show that the randomness of semi-functional components of ciphertext and keys will blind the message completely. We use the abbreviation ‘type’ to identify semi-functional type.

Suppose that after receiving challenge ciphertext $\overline{C}^* = (\overline{C}_0^*, \mathbf{C}^*)$, adversary modifies it to $\overline{C} = (\overline{C}_0^*, \mathbf{C}^*)$. Lemma 1 emphasizes that such a ciphertext \overline{C} can

pass Eq. (1) if and only if $\overline{C}_0^* = \overline{C}_0^* \cdot g_1^{\mathbf{B} \begin{pmatrix} 0 \\ \tau \end{pmatrix}}$ (for some $\tau \in \mathbb{Z}_p$). If adversary comes up with such \overline{C} , one can devise an efficient \mathcal{D}_d -MatDH solver (described in Footnote 3 in Lemma 2). Therefore we can assume that the adversary always query well-formed ciphertext to decrypt oracle. This fact plays a key role in Lemma 3.

Lemma 1. *Let $\overline{C} = (\overline{C}_0, \mathbf{C})$ be a ciphertext (possibly ill-formed). Then the ciphertext \overline{C} will satisfy Eq. (1) if and only if $\overline{C}_0 = g_1^{(\xi \mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1 + \mathbf{B} \begin{pmatrix} 0 \\ \tau \end{pmatrix}}$ and $C_1 = g_1^{c_1}$ for any $\tau \in \mathbb{Z}_p$.*

Proof. The sufficiency of this lemma follows from *associativity* and the relation $(\mathbf{I}_d \ 0) \mathbf{Z}^\top \mathbf{B} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \mathbf{0}$.

The necessary part of this lemma is given as follows. The RHS of Eq. (1) evaluates to $e(g_1, g_2)^{(\mathbf{I}_d \mathbf{0})\mathbf{Z}^\top(\xi\mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1}$. A satisfied verification requires the LHS to evaluate the same. The exponent of the G_T element computed in Eq. (1) can be expressed as a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{V}$ where $\mathbf{A} = (\mathbf{I}_d \mathbf{0})\mathbf{Z}^\top \in \mathbb{Z}_p^{d \times (d+1)}$, $\mathbf{x} \in \mathbb{Z}_p^{(d+1)}$ and $\mathbf{V} = (\mathbf{I}_d \mathbf{0})\mathbf{Z}^\top(\xi\mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1 \in \mathbb{Z}_p^d$. We can write $\mathbf{V} = \mathbf{A}\mathbf{x}'$ where $\mathbf{x}' = (\xi\mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1$, it simply implies that \mathbf{x}' is a solution of the system $\mathbf{A}\mathbf{x} = \mathbf{V}$.

Suppose there exists a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{V}$ where $\mathbf{A} \in \mathbb{Z}_p^{m \times n}$, $\mathbf{x} \in \mathbb{Z}_p^n$ and $\mathbf{V} \in \mathbb{Z}_p^m$ such that $\text{Rank}(\mathbf{A}) = r \in \mathbb{N}$. We define the solution set of such linear system to be $S = \{\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{V}\}$ and the solution of corresponding homogeneous equations is $S_0 = \{\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{0}\}$. Naturally, if a solution $\mathbf{x}' \in S$ is available, then $S = \{\mathbf{x}' + \mathbf{x} : \mathbf{x} \in S_0\}$. Due to rank-nullity theorem, $n = \text{Rank}(\mathbf{A}) + \dim(S_0)$. Therefore $\dim(S_0) = n - r$.

Here, in case of Eq. (1), we see that $r = \text{Rank}(\mathbf{A}) = d$ as $\mathbf{A} = (\mathbf{I}_d \mathbf{0})\mathbf{Z}^\top$ where $\mathbf{Z} \in \mathbb{Z}_p^{(d+1) \times (d+1)}$ is invertible and $n = (d + 1)$. Therefore $\dim(S_0) = 1$. That means there exists non-trivial $\mathbf{x}_0 \in S_0$ and it spans the space S_0 alone. Now due to our construction, $(\mathbf{I}_d \mathbf{0})\mathbf{Z}^\top \mathbf{B} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \mathbf{0}$. Therefore $\mathbf{x}_0 = \mathbf{B} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is a solution of homogeneous equation. As $\dim(S_0) = 1$, clearly $\{\mathbf{x}_0\}$ is the basis of S_0 . Thus $S_0 = \left\{ \mathbf{B} \begin{pmatrix} 0 \\ \tau \end{pmatrix} : \tau \in \mathbb{Z}_p \right\}$. Therefore $S = \left\{ (\xi\mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1 + \mathbf{B} \begin{pmatrix} 0 \\ \tau \end{pmatrix} : \tau \in \mathbb{Z}_p \right\}$. \square

Theorem 1. *Suppose a regular decryption pair encoding scheme P for predicate R is both SMH-Secure and CMH-Secure¹ in \mathcal{G} , and the \mathcal{D}_d -Matrix DH Assumption holds in \mathcal{G} . Then the scheme Π_R is fully CCA-secure encryption scheme if \mathcal{H} is collision resistant hash function. More precisely, for any PPT adversary \mathcal{A} that makes at most q_1 key queries before challenge, at most q_2 key queries after challenge and at most Q decryption queries throughout the game, there exist PPT algorithms $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$ such that for any λ ,*

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\Pi_R}(\lambda) &\leq (2q_1 + 2Q + 3) \cdot \text{Adv}_{\mathcal{B}_1}^{\mathcal{D}_d\text{-MatDH}}(\lambda) + q_1 \cdot \text{Adv}_{\mathcal{B}_2}^{\text{CMH}}(\lambda) + \text{Adv}_{\mathcal{B}_3}^{\text{SMH}}(\lambda) \\ &\quad + Q \cdot \text{Adv}_{\mathcal{B}_4}^{\text{CRH}}(\lambda). \end{aligned}$$

We give a hybrid argument to prove Theorem 1. Probabilistic polynomial time adversary \mathcal{A} is capable of making at most q_1 key queries before challenge phase, at most q_2 key queries after challenge phase and at most Q decryption queries throughout the game.

Game_0 is the real security game and Game_4 is the game where all secret keys are type-3 semi-functional keys, all altKeys are type-3 semi-functional altKeys and the challenge ciphertext is semi-functional ciphertext of random message (therefore is independent of the message that is to be encrypted). The indistinguishability of Game_0 and Game_4 is proven via the sequence of games of Table 2.

The idea is to change each game only by a small margin and prove indistinguishability of two consecutive games. First we make the challenge

¹ Here SMH means (1, poly)-SMH and CMH means (1, 1)-CMH (see [3, 5]).

Table 2. Outline of proof strategy

Games	Difference from previous game	Indistinguishability from previous game
Game_0	-	-
Game_1	Challenge ciphertext is semi-functional	[5]
$\text{Game}_{2,i-1,3}$	All the $(i - 1)$ secret keys are type-3 key ($i \leq q_1$)	[5]
$\text{Game}_{2,i,1}$	i^{th} secret key is type-1 key ($i \leq q_1$)	[5]
$\text{Game}_{2,i,2}$	i^{th} secret key is type-2 key ($i \leq q_1$)	[5]
$\text{Game}_{2,i,3}$	i^{th} secret key is type-3 key ($i \leq q_1$)	[5]
$\text{Game}_{2,q_1+1,1}$	All post-challenge secret keys are type-1 key	[5]
$\text{Game}_{2,q_1+1,2}$	All post-challenge secret keys are type-2 key	[5]
$\text{Game}_{2,q_1+1,3}$	All post-challenge secret keys are type-3 key	[5]
$\text{Game}_{3,i-1,3}$	All the $(i - 1)$ altKeys are type-3 altKey ($i \leq Q$)	Lemma 2
$\text{Game}_{3,i,1}$	i^{th} altKey is type-1 altKey ($i \leq Q$)	Lemma 2
$\text{Game}_{3,i,2}$	i^{th} altKey is type-2 altKey ($i \leq Q$)	Lemma 3
$\text{Game}_{3,i,3}$	i^{th} altKey is type-3 altKey ($i \leq Q$)	Lemma 4
Game_4	Challenge ciphertext is semi-functional encryption of a random message	Lemma 5

ciphertext semi-functional. Then we modify each i^{th} pre-challenge key to type- j semi-functional key in $\text{Game}_{2,i,j}$ for each $i \in [q_1]$ and $j \in \{1, 2, 3\}$. Note that to answer i^{th} pre-challenge key query, the simulator chooses fresh $\beta_i \xleftarrow{\$} \mathbb{Z}_p$. Then we modify all the post-challenge keys to type- j keys together in $\text{Game}_{2,q_1+1,j}$ for each $i \in [q_1 + 1, q]$ and $j \in \{1, 2, 3\}$. Here, however, the simulator uses same $\beta \xleftarrow{\$} \mathbb{Z}_p$ to answer every post-challenge key query. Then we modify each i^{th} altKey to type- j semi-functional altKey in $\text{Game}_{3,i,j}$ for each $i \in [Q]$ and $j \in \{1, 2, 3\}$. Note that the simulator uses same $\eta \xleftarrow{\$} \mathbb{Z}_p$ to compute all the altKeys. In the final game Game_4 , we show that the ciphertext is completely independent of \mathbf{b} . Therefore the advantage of adversary \mathcal{A} in Game_4 is 0. Note that Game_1 and $\text{Game}_{2,q_1+1,3}$ are also denoted by $\text{Game}_{2,0,3}$ and $\text{Game}_{3,0,3}$ respectively.

As mentioned in Table 2, we have used the proof technique of [5] to argue indistinguishability of several games. However, the games that *deal with* changes in altKey and the final game are primary contributions of this work. We, however, have included the description of games that we have mimicked (and modified for our requirement) from [5] in the full version [19]. Here we concentrate only in $\text{Game}_{3,i,1}$, $\text{Game}_{3,i,2}$, $\text{Game}_{3,i,3}$ for $i \in [Q]$ and Game_4 .

Note that in $\text{Game}_{2,q_1+1,3}$, the challenge ciphertext is semi-functional and all the secret keys are type-3 semi-functional. However, all the altKeys at this point are normal. We then change the altKeys to type-3 semi-functional altKey one by one. For every $i \in [Q]$, this is done via changing normal altKey to type-1 altKey first in $\text{Game}_{3,i,1}$. Subsequently we change it into type-2 altKey in $\text{Game}_{3,i,2}$ and to type-3 altKey in $\text{Game}_{3,i,3}$. In $\text{Game}_{3,i,2}$ we introduce the randomness η that hides the master secret key in the final game. This effectively allows us to show that in the final game, the simulator can simulate all the secret keys and the altKeys properly and the challenge ciphertext is semi-functional ciphertext of random message.

Note that in all of these above mentioned games, the decryption query can only be made on ciphertext $\bar{\mathbf{C}}$ where $\bar{\mathbf{C}}_0 = g_1^{(\xi \mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1}$ and $\bar{\mathbf{C}}_1 = g_1^{c_1}$. The reason is discussed in Footnote 3 in Lemma 2.

3.3.1 Semi-functional Algorithms

Following semi-functional algorithms will be used in the security proof.

- $\text{SFSetup}(1^\lambda, \kappa)$: It runs $(mpk, msk) \leftarrow \text{Setup}(1^\lambda, \kappa)$. Additionally it outputs \widehat{mpk}_{base} , \widehat{mpk}_b and \widehat{mpk}_z where $\widehat{mpk}_{base} = g_2^{\mathbf{Z}(\frac{\mathbf{0}}{1})}$, $\widehat{mpk}_b = \left(e(g_1, g_2) \boldsymbol{\alpha}^\top \mathbf{B}(\frac{\mathbf{0}}{1}), g_1^{\mathbf{B}(\frac{\mathbf{0}}{1})}, g_1^{\mathbf{W}_1 \mathbf{B}(\frac{\mathbf{0}}{1})}, \dots, g_1^{\mathbf{W}_{n+2} \mathbf{B}(\frac{\mathbf{0}}{1})} \right)$ and $\widehat{mpk}_z = \left(g_2^{\mathbf{W}_1^\top \mathbf{Z}(\frac{\mathbf{0}}{1})}, \dots, g_2^{\mathbf{W}_{n+2}^\top \mathbf{Z}(\frac{\mathbf{0}}{1})} \right)$.
- $\text{SFKeyGen}(x, msk, \widehat{mpk}_z, \widehat{mpk}_{base}, \text{type}, \beta)$: Runs $(\mathbf{k}; m_2) \leftarrow \text{EncK}(x, N)$. Chooses $\mathbf{r}_1, \dots, \mathbf{r}_{m_2} \xleftarrow{\$} \mathbb{Z}_p^d$ and $\hat{r}_1, \dots, \hat{r}_{m_2} \xleftarrow{\$} \mathbb{Z}_p$. It defines $\mathbf{R} = \left(\left(\begin{smallmatrix} \mathbf{r}_1 \\ 0 \end{smallmatrix} \right), \dots, \left(\begin{smallmatrix} \mathbf{r}_{m_2} \\ 0 \end{smallmatrix} \right) \right) \in \left(\mathbb{Z}_p^{(d+1)} \right)^{m_2}$ and $\widehat{\mathbf{R}} = \left(\left(\begin{smallmatrix} \mathbf{0} \\ \hat{r}_1 \end{smallmatrix} \right), \dots, \left(\begin{smallmatrix} \mathbf{0} \\ \hat{r}_{m_2} \end{smallmatrix} \right) \right) \in \left(\mathbb{Z}_p^{(d+1)} \right)^{m_2}$. Outputs the secret key

$$\mathbf{K} = \begin{cases} g_2^{\mathbf{k}(\boldsymbol{\alpha}, \mathbf{R}, \mathbb{W}) + \mathbf{k}(\mathbf{0}, \widehat{\mathbf{R}}, \mathbb{W})} & \text{if type} = 1 \\ g_2^{\mathbf{k}(\boldsymbol{\alpha}, \mathbf{R}, \mathbb{W}) + \mathbf{k}(\mathbf{Z}(\frac{\mathbf{0}}{\beta}), \widehat{\mathbf{R}}, \mathbb{W})} & \text{if type} = 2 \\ g_2^{\mathbf{k}(\boldsymbol{\alpha}, \mathbf{R}, \mathbb{W}) + \mathbf{k}(\mathbf{Z}(\frac{\mathbf{0}}{\beta}), \mathbf{0}, \mathbb{W})} & \text{if type} = 3 \end{cases}$$

where $\mathbf{k}(\boldsymbol{\alpha}, \mathbf{R}, \mathbb{W}) + \mathbf{k}(\mathbf{Z}(\frac{\mathbf{0}}{\beta}), \widehat{\mathbf{R}}, \mathbb{W}) =$

$$\left\{ b_\iota \boldsymbol{\alpha} + b_\iota \mathbf{Z}(\frac{\mathbf{0}}{\beta}) + \sum_{j \in [m_2]} b_{\iota j} \mathbf{Z}(\frac{\mathbf{r}_j}{\hat{r}_j}) + \sum_{\substack{j \in [m_2] \\ k \in [n]}} b_{\iota j k} \mathbf{W}_k^\top \mathbf{Z}(\frac{\mathbf{r}_j}{\hat{r}_j}) \right\}_{\iota \in [m_1]}.$$

- $\text{SFEncrypt}(y, M, mpk, \widehat{mpk}_b)$: It runs $(\mathbf{c}; w_2) \leftarrow \text{EncC}(y, N)$. Chooses $\mathbf{s}_0, \dots, \mathbf{s}_{w_2} \xleftarrow{\$} \mathbb{Z}_p^d$ and $\hat{s}_0, \dots, \hat{s}_{w_2} \xleftarrow{\$} \mathbb{Z}_p$. Then it defines $\mathbf{S} = \left(\left(\begin{smallmatrix} \mathbf{s}_0 \\ 0 \end{smallmatrix} \right), \dots, \left(\begin{smallmatrix} \mathbf{s}_{w_2} \\ 0 \end{smallmatrix} \right) \right) \in \left(\mathbb{Z}_p^{(d+1)} \right)^{(w_2+1)}$ and $\widehat{\mathbf{S}} = \left(\left(\begin{smallmatrix} \mathbf{0} \\ \hat{s}_0 \end{smallmatrix} \right), \dots, \left(\begin{smallmatrix} \mathbf{0} \\ \hat{s}_{w_2} \end{smallmatrix} \right) \right) \in \left(\mathbb{Z}_p^{(d+1)} \right)^{(w_2+1)}$. It computes the semi-functional ciphertext $\mathbf{C} = (C_1, \dots, C_{w_1}, C_{w_1+1})$ where for

$\tilde{t} \in [w_1]$, $C_{\tilde{t}} = g_1^{\mathbf{c}_{\tilde{t}}(\mathbf{S}, \mathbb{W}) + \mathbf{c}_{\tilde{t}}(\widehat{\mathbf{S}}, \mathbb{W})} = g_1$
 and $C_{w_1+1} = M \cdot e(g_1, g_2)^{\boldsymbol{\alpha}^\top \mathbf{B} \begin{pmatrix} s_0 \\ \hat{s}_0 \end{pmatrix}}$. Then it computes $\xi = \mathcal{H}(\mathbf{C})$ and outputs $\overline{\mathbf{C}} = (\overline{\mathbf{C}}_0, \mathbf{C})$.

- **SFAltKeyGen**($\overline{\mathbf{C}}, x, msk, \widehat{mpk}_z, \widehat{mpk}_{base}, \text{type}, \eta$): Runs $(\mathbf{k}; m_2) \leftarrow \text{EncK}(x, N)$ and $\mathbf{E} \leftarrow \text{Pair}(x, y, N)$. Chooses $\mathbf{r}_1, \dots, \mathbf{r}_{m_2}, \mathbf{t} \xleftarrow{\$} \mathbb{Z}_p^d$ and $\hat{\mathbf{t}} \xleftarrow{\$} \mathbb{Z}_p$. Then it defines $\mathbf{R} = \left(\begin{pmatrix} \mathbf{r}_1 \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} \mathbf{r}_{m_2} \\ 0 \end{pmatrix} \right) \in \left(\mathbb{Z}_p^{(d+1)} \right)^{m_2}$.

Then the normal key is $\mathbf{K} = \left\{ g_2^{k_\iota(\boldsymbol{\alpha}, \mathbf{R}, \mathbb{W})} \right\}_{\iota \in [m_1]} \in (G_2^{(d+1)})^{m_1}$ where
 $k_\iota(\boldsymbol{\alpha}, \mathbf{R}, \mathbb{W}) = b_\iota \boldsymbol{\alpha} + \sum_{j \in [m_2]} b_{\iota j} \mathbf{Z} \begin{pmatrix} \mathbf{r}_j \\ 0 \end{pmatrix} + \sum_{\substack{j \in [m_2] \\ k \in [n]}} b_{\iota j k} \mathbf{W}_k^\top \mathbf{Z} \begin{pmatrix} \mathbf{r}_j \\ 0 \end{pmatrix}$ for $\iota \in [m_1]$.

Then it computes $(\tilde{\mathbf{K}}_1, \dots, \tilde{\mathbf{K}}_{w_1})$ where $\tilde{\mathbf{K}}_{\tilde{t}} = \prod_{\iota \in [m_1]} (\mathbf{K}_\iota)^{\mathbf{E}_{\iota \tilde{t}}}$ for $\tilde{t} \in [w_1]$.

Defines modified key $\hat{\mathbf{K}} = (\hat{\mathbf{K}}_0, \Phi, \tilde{\mathbf{K}}_1, \tilde{\mathbf{K}}_2, \dots, \tilde{\mathbf{K}}_{w_1})$ where

$$(\hat{\mathbf{K}}_0, \Phi) = \begin{cases} \left(\begin{pmatrix} \mathbf{z} \begin{pmatrix} \mathbf{t} \\ \hat{\mathbf{t}} \end{pmatrix} \\ g_2 \end{pmatrix}, g_2^{(\xi \mathbf{W}_{n+1}^\top + \mathbf{W}_{n+2}^\top) \mathbf{z} \begin{pmatrix} \mathbf{t} \\ \hat{\mathbf{t}} \end{pmatrix}} \right) & \text{if type} = 1 \\ \left(\begin{pmatrix} \mathbf{z} \begin{pmatrix} \mathbf{t} \\ \hat{\mathbf{t}} \end{pmatrix} \\ g_2 \end{pmatrix}, g_2^{\mathbf{z} \begin{pmatrix} \mathbf{0} \\ \eta u \end{pmatrix} + (\xi \mathbf{W}_{n+1}^\top + \mathbf{W}_{n+2}^\top) \mathbf{z} \begin{pmatrix} \mathbf{t} \\ \hat{\mathbf{t}} \end{pmatrix}} \right) & \text{if type} = 2 \\ \left(\begin{pmatrix} \mathbf{z} \begin{pmatrix} \mathbf{t} \\ 0 \end{pmatrix} \\ g_2 \end{pmatrix}, g_2^{\mathbf{z} \begin{pmatrix} \mathbf{0} \\ \eta u \end{pmatrix} + (\xi \mathbf{W}_{n+1}^\top + \mathbf{W}_{n+2}^\top) \mathbf{z} \begin{pmatrix} \mathbf{t} \\ 0 \end{pmatrix}} \right) & \text{if type} = 3 \end{cases}$$

for $u = \sum_{\iota \in [m_1]} b_\iota \mathbf{E}_{\iota 1}$ and $\xi = \mathcal{H}(\mathbf{C})$ in case of given ciphertext $\overline{\mathbf{C}} = (\overline{\mathbf{C}}_0, \mathbf{C})$.

3.3.2 Sequence of Games

Here we present indistinguishability of $\text{Game}_{3,i,1}$, $\text{Game}_{3,i,2}$, $\text{Game}_{3,i,3}$ and Game_4 for $1 \leq i \leq Q$ of Table 2 in the following lemmas.

Lemma 2 ($\text{Game}_{3,i-1,3}$ to $\text{Game}_{3,i,1}$). *For $i \in [Q]$, for any efficient adversary \mathcal{A} that makes at most q key queries and at most Q decryption queries, there exists a PPT algorithm \mathcal{B}_1 such that $|\text{Adv}_{\mathcal{A}}^{3,i-1,3}(\lambda) - \text{Adv}_{\mathcal{A}}^{3,i,1}(\lambda)| \leq \text{Adv}_{\mathcal{B}_1}^{\mathcal{D}_d\text{-MatDH}}(\lambda)$.*

Proof. The algorithm \mathcal{B}_1 gets input $(\mathbf{G}, g_2^\mathbf{T}, g_2^{\mathbf{T} \begin{pmatrix} \mathbf{y} \\ \hat{\mathbf{y}} \end{pmatrix}})$ as $\mathcal{D}_d\text{-MatDH}$ problem instance where $\hat{\mathbf{y}} = 0$ or $\hat{\mathbf{y}} \xleftarrow{\$} \mathbb{Z}_p$ and $\mathbf{T} \xleftarrow{\$} \mathcal{D}_d, \mathbf{y} \xleftarrow{\$} \mathbb{Z}_p^d$.

Setup. \mathcal{B}_1 chooses $\tilde{\mathbf{B}} \xleftarrow{\$} \mathbb{G}\mathbb{L}_{p,d+1}$, $\mathbf{J} \xleftarrow{\$} \mathbb{G}\mathbb{L}_{p,d}$ and sets $\mathbf{B} = \tilde{\mathbf{B}} \begin{pmatrix} \mathbf{I}_d & \mathbf{M}^{-\top} \mathbf{c}^\top \\ \mathbf{0} & -1 \end{pmatrix}$ and $\mathbf{D} = \begin{pmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}$ where $\mathbf{T} = \begin{pmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{c} & 1 \end{pmatrix}$ due to \mathcal{D}_d -MatDH assumption. Then it defines $\mathbf{Z} = \mathbf{B}^{-\top} \mathbf{D} = \tilde{\mathbf{B}}^{-\top} \begin{pmatrix} \mathbf{I}_d & \mathbf{0} \\ \mathbf{c} \mathbf{M}^{-1} & -1 \end{pmatrix} \begin{pmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix} = \tilde{\mathbf{B}}^{-\top} \mathbf{T} \begin{pmatrix} \mathbf{J} & \mathbf{0} \\ \mathbf{0} & -1 \end{pmatrix}$. It then defines $\tilde{\mathbf{Z}} = \begin{pmatrix} \mathbf{J} & \mathbf{0} \\ \mathbf{0} & -1 \end{pmatrix}$ so that $\mathbf{Z} = \tilde{\mathbf{B}}^{-\top} \mathbf{T} \tilde{\mathbf{Z}}$. \mathcal{B}_1 therefore can compute the public parameters as $g_1^{\mathbf{B} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}} = g_1^{\tilde{\mathbf{B}} \begin{pmatrix} \mathbf{I}_d \\ \mathbf{0} \end{pmatrix}}$ and $g_2^{\mathbf{Z}} = g_2^{\tilde{\mathbf{B}}^{-\top} \mathbf{T} \tilde{\mathbf{Z}}}$. Then \mathcal{B}_1 chooses $\alpha \xleftarrow{\$} \mathbb{Z}_p^{(d+1)}$ and $\mathbf{W} = (\mathbf{W}_1, \dots, \mathbf{W}_{n+2}) \xleftarrow{\$} \left(\mathbb{Z}_p^{(d+1) \times (d+1)} \right)^{(n+2)}$ and publishes public key mpk . Note that \mathcal{B}_1 cannot compute \widehat{mpk}_b but can compute \widehat{mpk}_z as it can compute \widehat{mpk}_{base} . It chooses $\beta, \eta \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random.

Key Queries. On j^{th} secret key query x ($j \leq q_1$), outputs type-3 secret key $\mathbf{K} \leftarrow \text{SFKeyGen}(x, msk, -, \widehat{mpk}_{base}, 3, \beta_j)$ after choosing $\beta_j \xleftarrow{\$} \mathbb{Z}_p$.

Dec Queries. On j^{th} decryption query $(x, \overline{\mathbf{C}})$ where $\overline{\mathbf{C}}$ is a ciphertext on data-index y , if $R(x, y) \neq 1$, aborts. Otherwise \mathcal{B}_1 computes altKey $\hat{\mathbf{K}}$ and returns $\text{AltDecrypt}(\overline{\mathbf{C}}, \hat{\mathbf{K}})$ to \mathcal{A} . Here we emphasize that the decryption queries will follow a certain structure given in the footnote². We now describe the altKey generation procedure.

- If $j > i$, it is normal altKey. As \mathcal{B}_1 knows msk , it computes the altKey $\hat{\mathbf{K}} \leftarrow \text{AltKeyGen}(\overline{\mathbf{C}}, x, msk)$.
- If $j < i$, it is type-3 semi-functional altKey. \mathcal{B}_1 computes type-3 altKey $\hat{\mathbf{K}} \leftarrow \text{SFAltKeyGen}(\overline{\mathbf{C}}, x, msk, -, \widehat{mpk}_{base}, 3, \eta)$.
- If $j = i$, it runs $(\mathbf{k} = (k_1, \dots, k_{m_1}); m_2) \leftarrow \text{EncK}(x, N)$ and $\mathbf{E} \leftarrow \text{Pair}(x, y, N)$. Chooses $\mathbf{r}_1, \dots, \mathbf{r}_{m_2} \xleftarrow{\$} \mathbb{Z}_p^d$ and defines $\mathbf{R} = \left(\begin{pmatrix} \mathbf{r}_1 \\ \mathbf{0} \end{pmatrix}, \dots, \begin{pmatrix} \mathbf{r}_{m_2} \\ \mathbf{0} \end{pmatrix} \right)$. It generates normal key $\mathbf{K} = (K_1, \dots, K_{m_1})$ where for each $\iota \in [m_1]$, $K_\iota = g_2^{k_\iota(\alpha, \mathbf{R}, \mathbf{W})} = g_2^{\left(b_\iota \alpha + \sum_{j \in [m_2]} b_{\iota j} \mathbf{z} \begin{pmatrix} \mathbf{r}_j \\ \mathbf{0} \end{pmatrix} + \sum_{k \in [n]} b_{\iota j k} \mathbf{W}_k^\top \mathbf{z} \begin{pmatrix} \mathbf{r}_j \\ \mathbf{0} \end{pmatrix} \right)}$. It then computes $(\tilde{K}_1, \dots, \tilde{K}_{w_1})$

² Suppose given ciphertext is $\overline{\mathbf{C}} = (\overline{\mathbf{C}}_0, \mathbf{C})$ where $\overline{\mathbf{C}}_0 = g_1^{(\xi \mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1 + \mathbf{B} \begin{pmatrix} \mathbf{0} \\ \tau \end{pmatrix}}$ for some $\tau \in \mathbb{Z}_p$ and $C_1 = g_1^{c_1}$. Note that it satisfies the verification in Eq. (1) as can be seen in Lemma 1. However, as the simulator knows \mathbf{W}_{n+1} and \mathbf{W}_{n+2} , it can compute

$L = g_1^{(\xi \mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1}$. Therefore it gets hold of $g_1^{\mathbf{B} \begin{pmatrix} \mathbf{0} \\ \tau \end{pmatrix}}$ by computing $\overline{\mathbf{C}}_0/L$. Since, \mathbf{B} and \mathbf{Z} are simulated exactly as Lemma 2 (see the **Setup** of Lemma 2), and \mathcal{B}_1 implicitly sets $\tilde{\mathbf{Z}}^{-1} \begin{pmatrix} \mathbf{y} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{r} \\ \mathbf{r} \end{pmatrix}$ to compute i^{th} altKey, $e \left(g_1^{\mathbf{B} \begin{pmatrix} \mathbf{0} \\ \tau \end{pmatrix}}, g_2^{\mathbf{z} \begin{pmatrix} \mathbf{r} \\ \mathbf{r} \end{pmatrix}} \right)$ evaluation

will allow the simulator to decide the \mathcal{D}_d -MatDH problem instance. Thus, under \mathcal{D}_d -MatDH assumption, the adversary can't make such decryption query. Therefore any decryption query \mathcal{A} makes, to satisfy Eq. (1), the queried ciphertext $\overline{\mathbf{C}}$ must follow the relation that $\overline{\mathbf{C}}_0 = g_1^{(\xi \mathbf{W}_{n+1} + \mathbf{W}_{n+2})c_1}$ and $C_1 = g_1^{c_1}$ where $\xi = \mathcal{H}(\mathbf{C})$.

where $\tilde{K}_{\tilde{i}} = \prod_{\iota \in [m_1]} (K_{\iota})^{\mathbf{E}^{\iota \tilde{i}}}$ for each $\tilde{i} \in [w_1]$.

Given $\overline{\mathbf{C}} = (\overline{\mathbf{C}}_0, \mathbf{C})$, \mathcal{B}_1 computes $\xi = \mathcal{H}(\mathbf{C})$. To compute the altKey, it implicitly sets $\tilde{\mathbf{Z}}^{-1} \begin{pmatrix} \mathbf{y} \\ \hat{y} \end{pmatrix} = \begin{pmatrix} \mathbf{r} \\ \hat{r} \end{pmatrix}$. Therefore $g_2 \begin{pmatrix} \mathbf{r} \\ \hat{r} \end{pmatrix} = g_2 \tilde{\mathbf{B}}^{-\top} \mathbf{T} \begin{pmatrix} \mathbf{y} \\ \hat{y} \end{pmatrix}$. The simulator then computes modified key $\hat{K} = (\hat{K}_0, \Phi \cdot \tilde{K}_1, \tilde{K}_2, \dots, \tilde{K}_{w_1})$ where $\hat{K}_0 = g_2 \begin{pmatrix} \mathbf{r} \\ \hat{r} \end{pmatrix}$, $\Phi = g_2^{(\xi \mathbf{W}_{n+1}^{\top} + \mathbf{W}_{n+2}^{\top})} \mathbf{z} \begin{pmatrix} \mathbf{r} \\ \hat{r} \end{pmatrix}$ and therefore is efficiently computable. It is evident from the description that if $\hat{y} = 0$, the key is a normal altKey whereas if $\hat{y} \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, the key is type-1 altKey.

Challenge. On receiving the challenge (y^*, M_0, M_1) , \mathcal{B}_1 picks $\mathbf{b} \stackrel{\$}{\leftarrow} \{0, 1\}$. It runs $(\mathbf{c} = (c_1, \dots, c_{w_1}); w_2) \leftarrow \text{EncC}(y^*, N)$. For each $j \in [0, w_2]$ it chooses $\begin{pmatrix} s'_j \\ \hat{s}'_j \end{pmatrix} \stackrel{\$}{\leftarrow} \mathbb{Z}_p^{(d+1)}$ and implicitly sets $\begin{pmatrix} s_j \\ \hat{s}_j \end{pmatrix} = \mathbf{B}^{-1} \begin{pmatrix} s'_j \\ \hat{s}'_j \end{pmatrix}$. Then \mathcal{B}_1 computes \mathbf{C}^* as it knows $\alpha, \mathbf{W}_1, \dots, \mathbf{W}_{n+2}$. It evaluates $\xi^* = \mathcal{H}(\mathbf{C}^*)$ to compute $\overline{\mathbf{C}}^* = (\overline{\mathbf{C}}_0^*, \mathbf{C}^*)$ where $\overline{\mathbf{C}}_0^* = g_1^{(\xi^* \mathbf{W}_{n+1} + \mathbf{W}_{n+2})} \begin{pmatrix} s'_0 \\ \hat{s}'_0 \end{pmatrix}$ and outputs $\overline{\mathbf{C}}^*$.

Key Queries. Same as Phase-I key queries.

Dec Queries. Same as Phase-I dec queries.

Guess. \mathcal{A} halts with output \mathbf{b}' . \mathcal{B}_1 outputs 1 if $\mathbf{b}' = \mathbf{b}$ and 0 otherwise. \square

Lemma 3 (Game $_{3,i,1}$ to Game $_{3,i,2}$). *For $i \in [Q]$, for all adversary \mathcal{A} we have $|\text{Adv}_{\mathcal{A}}^{3,i,1}(\lambda) - \text{Adv}_{\mathcal{A}}^{3,i,2}(\lambda)| = 0$ if \mathcal{H} is Collision Resistant Hash Function.*

To prove the indistinguishability of the two games, we use modified SFSetup namely SFSetup' that introduces independent randomness in \widehat{mpk}_b and \widehat{mpk}_z by means of security property *parameter-hiding* [5, Lemma 2] which is undetectable to the adversary \mathcal{A} (see [19] for more details). Note that this newly introduced randomness does not affect the public key mpk . Then we show that introduction of such new randomness allows us to argue the indistinguishability. Recall that the challenge ciphertext is semi-functional and is denoted by $\overline{\mathbf{C}}^*$, the secret keys \mathbf{K} are all type-3 keys and the altKey, computed to answer i^{th} decryption query, is denoted by \hat{K} . To prove the lemma, note that, it is sufficient to argue that the joint distribution of semi-functional ciphertext, semi-functional secret keys and semi-functional altKeys stays identical, independent of if the altKey is type-1 altKey or a type-2 altKey.

Precisely, here we prove that joint distribution of $\{\mathbf{K}, \overline{\mathbf{C}}^*, \hat{K}\}$ if \hat{K} is type-1 altKey is identical to joint distribution of $\{\mathbf{K}, \overline{\mathbf{C}}^*, \hat{K}\}$ if \hat{K} is type-2 altKey. Note that $\overline{\mathbf{C}}^* = (\overline{\mathbf{C}}_0^*, \mathbf{C}^*)$ such that $\overline{\mathbf{C}}_0^* = (\xi^* \mathbf{W}_{n+1} + \mathbf{W}_{n+2}) \mathbf{B} \begin{pmatrix} s_0 \\ \hat{s}_0 \end{pmatrix} + \mathbf{B} \begin{pmatrix} \mathbf{0} \\ (\xi^* \hat{w}_{n+1} + \hat{w}_{n+2}) \hat{s}_0 \end{pmatrix}$ where $\xi^* = \mathcal{H}(\mathbf{C}^*)$. Now we prove

our claim that, the joint distributions of $\{\mathbf{K}, \overline{\mathbf{C}}^*, \hat{\mathbf{K}}\}$ behaves identically for both type-1 and type-2 altKey $\hat{\mathbf{K}}$.

Claim. The joint distribution of $\{\mathbf{K}, \overline{\mathbf{C}}^*, \hat{\mathbf{K}}\}$ if $\hat{\mathbf{K}}$ is type-1 altKey is identical to joint distribution of $\{\mathbf{K}, \overline{\mathbf{C}}^*, \hat{\mathbf{K}}\}$ if $\hat{\mathbf{K}}$ is type-2 altKey.

Proof. Note that \mathbf{K} is type-3 key in both the distributions and can be computed by the simulator as it knows msk and $\widehat{mpk}_{\text{base}}$. Due to linearity of pair encoding, the challenge ciphertext $\overline{\mathbf{C}}^*$ and the altKey $\hat{\mathbf{K}}$ can be expressed as product of normal component and semi-functional component. Since the simulator knows msk and can compute the normal components, it suffices to show that the joint distributions are identical if the joint distribution of semi-functional components of $\overline{\mathbf{C}}^*$ and $\hat{\mathbf{K}}$ are identically distributed.

Notice that due to the introduction of $\hat{\mathbf{w}}$ (see [19]), the semi-functional ciphertext component $\overline{\mathbf{C}}_0^{*'} and the term Φ' used in altKey, is affected. To prove our claim, it suffices to argue that the following two distributions $(\overline{\mathbf{C}}_0^{*'}, \Phi')$ are identically distributed:$

$$\left\{ g_1^{(\xi^* \mathbf{w}_{n+1} + \mathbf{w}_{n+2})\mathbf{B} \begin{pmatrix} \mathbf{0} \\ \hat{s}_0 \end{pmatrix} + \mathbf{B} \begin{pmatrix} \mathbf{0} \\ (\xi^* \hat{\mathbf{w}}_{n+1} + \hat{\mathbf{w}}_{n+2})\hat{s}_0 \end{pmatrix}}, \right. \\ \left. g_2^{\mathbf{z} \begin{pmatrix} \mathbf{0} \\ (\xi \hat{\mathbf{w}}_{n+1} + \hat{\mathbf{w}}_{n+2})\hat{\mathbf{t}} \end{pmatrix} + (\xi \mathbf{w}_{n+1}^\top + \mathbf{w}_{n+2}^\top)\mathbf{z} \begin{pmatrix} \mathbf{0} \\ \hat{\mathbf{t}} \end{pmatrix}} \right\} \\ \left\{ g_1^{(\xi^* \mathbf{w}_{n+1} + \mathbf{w}_{n+2})\mathbf{B} \begin{pmatrix} \mathbf{0} \\ \hat{s}_0 \end{pmatrix} + \mathbf{B} \begin{pmatrix} \mathbf{0} \\ (\xi^* \hat{\mathbf{w}}_{n+1} + \hat{\mathbf{w}}_{n+2})\hat{s}_0 \end{pmatrix}}, \right. \\ \left. g_2^{\mathbf{z} \begin{pmatrix} \mathbf{0} \\ u\eta \end{pmatrix} + \mathbf{z} \begin{pmatrix} \mathbf{0} \\ (\xi \hat{\mathbf{w}}_{n+1} + \hat{\mathbf{w}}_{n+2})\hat{\mathbf{t}} \end{pmatrix} + (\xi \mathbf{w}_{n+1}^\top + \mathbf{w}_{n+2}^\top)\mathbf{z} \begin{pmatrix} \mathbf{0} \\ \hat{\mathbf{t}} \end{pmatrix}} \right\}.$$

By natural restriction $\overline{\mathbf{C}}^* \neq \overline{\mathbf{C}}$ where $\overline{\mathbf{C}}^*$ is challenge ciphertext and $\overline{\mathbf{C}}$ is ciphertext provided for decryption. Therefore $(\overline{\mathbf{C}}_0^*, \mathbf{C}^*) \neq (\overline{\mathbf{C}}_0, \mathbf{C})$.

Then any of the following two cases can happen,

1. $\overline{\mathbf{C}}_0^* \neq \overline{\mathbf{C}}_0$ and $\mathbf{C}^* = \mathbf{C}$: we show that such a case can't happen. Since $\mathbf{C}^* = \mathbf{C}$, $\xi^* = \xi$ and $\mathbf{C}_1 = \mathbf{C}_1^* = g_1^{c_1^*}$ naturally. This implies $\overline{\mathbf{C}}_0 = g_1^{(\xi^* \mathbf{w}_{n+1} + \mathbf{w}_{n+2})c_1^*} = \overline{\mathbf{C}}_0^*$ which is a contradiction.
2. $\mathbf{C}^* \neq \mathbf{C}$: the inequality $\mathbf{C}^* \neq \mathbf{C}$ implies $\xi^* \neq \xi$ (due to collision resistance of \mathcal{H}). Therefore $\xi^* \hat{\mathbf{w}}_{n+1} + \hat{\mathbf{w}}_{n+2}$ and $\xi \hat{\mathbf{w}}_{n+1} + \hat{\mathbf{w}}_{n+2}$ are pairwise independent as $\hat{\mathbf{w}}_{n+1}$ and $\hat{\mathbf{w}}_{n+2}$ are chosen uniformly at random. It implies that the semi-functional components of the ciphertext and altKey in $\text{Game}_{3,i,1}$ and $\text{Game}_{3,i,2}$ are identically distributed. \square

Lemma 4 ($\text{Game}_{3,i,2}$ to $\text{Game}_{3,i,3}$). *For $i \in [Q]$, for any efficient adversary \mathcal{A} that makes at most q key queries and at most Q decryption queries, there exists a PPT algorithm \mathcal{B}_1 such that $|\text{Adv}_{\mathcal{A}}^{3,i,2}(\lambda) - \text{Adv}_{\mathcal{A}}^{3,i,3}(\lambda)| \leq \text{Adv}_{\mathcal{B}_1}^{\mathcal{D}_a\text{-MatDH}}(\lambda)$.*

Proof. The algorithm \mathcal{B}_1 gets input $(\mathbf{G}, g_2^{\mathbf{T}}, g_2^{\mathbf{T}} \begin{pmatrix} \mathbf{y} \\ \hat{y} \end{pmatrix})$ as \mathcal{D}_d -MatDH problem instance where $\hat{y} = 0$ or $\hat{y} \xleftarrow{\$} \mathbb{Z}_p$ and $\mathbf{T} \xleftarrow{\$} \mathcal{D}_d$, $\mathbf{y} \xleftarrow{\$} \mathbb{Z}_p^d$.

The simulator description is same as Lemma 2 except while answering i^{th} decryption query. Here the altKey component $\hat{K}_1 = \Phi \cdot \hat{K}_1$ where Φ is now multiplied by $g_2^{\mathbf{z}} \begin{pmatrix} \mathbf{0} \\ \eta u \end{pmatrix} \in \mathbb{H}$ where $u = \sum_{\iota \in [m_1]} b_\iota \mathbf{E}_{\iota,1}$. As \mathcal{B}_1 knows $\widehat{mpk}_{\text{base}}$, it chooses $\eta \xleftarrow{\$} \mathbb{Z}_p$ to perform the simulation. In the similar light of Lemma 2, we see that if $\hat{y} = 0$, the altKey is a type-3 altKey whereas if $\hat{y} \xleftarrow{\$} \mathbb{Z}_p$, it is type-2 altKey. \square

Lemma 5 (Game_{3,Q,3} to Game₄). *For any adversary \mathcal{A} , we have $|\text{Adv}_{\mathcal{A}}^{3,Q,3}(\lambda) - \text{Adv}_{\mathcal{A}}^4(\lambda)| = 0$.*

Proof. As $\mathbf{Z} \in \mathbb{GL}_{p,d+1}$, one can express α as a linear combination of column vectors of \mathbf{Z} i.e. $\alpha = \mathbf{Z} \begin{pmatrix} \delta \\ \hat{\delta} \end{pmatrix}$ for $\delta \in \mathbb{Z}_p^d$ and $\hat{\delta} \in \mathbb{Z}_p$. In all the secret keys, $\hat{\delta}$ is hidden by uniformly random β_i (in case of pre-challenge secret key queries) and by uniformly random β (in case of post-challenge key queries). Note that in case of altKeys, the presence of α is limited only to \hat{K}_1 due to regular decryption property of pair encoding (precisely Property $\mathcal{P}5$) in the form of $u\alpha$ where $u = \sum_{\iota \in [m_1]} b_\iota \mathbf{E}_{\iota,1}$. The term, $u\mathbf{Z} \begin{pmatrix} \mathbf{0} \\ \eta \end{pmatrix}$, appears in the exponent of Φ of type-3 altKeys.

Therefore in all altKeys, $\hat{\delta}$ of α will be is hidden by uniformly random η .

Therefore we can replace $\hat{\delta}$ by $\hat{\delta} + t$ for $t \xleftarrow{\$} \mathbb{Z}_p$. Notice that such a change will affect the ciphertext in only one component namely $C_{w_1+1}^*$. The resultant $C_{w_1+1}^*$ will be $M_b \cdot e(g_1, g_2)^{\alpha^{\top} \mathbf{B} \begin{pmatrix} s_0 \\ \hat{s}_0 \end{pmatrix}} = M_b \cdot e(g_1, g_2)^{(\delta^{\top} \hat{\delta} + t) \mathbf{Z}^{\top} \mathbf{B} \begin{pmatrix} s_0 \\ \hat{s}_0 \end{pmatrix}} = M_b \cdot e(g_1, g_2)^{\alpha^{\top} \mathbf{B} \begin{pmatrix} s_0 \\ \hat{s}_0 \end{pmatrix}} \cdot e(g_1, g_2)^{(0 \ t) \mathbf{Z}^{\top} \mathbf{B} \begin{pmatrix} s_0 \\ \hat{s}_0 \end{pmatrix}} = M_b \cdot e(g_1, g_2)^{\alpha^{\top} \mathbf{B} \begin{pmatrix} s_0 \\ \hat{s}_0 \end{pmatrix}} \cdot e(g_1, g_2)^{t \hat{s}_0}$. Therefore $C_{w_1+1}^*$ encrypts $M_b \cdot e(g_1, g_2)^{t \hat{s}_0}$ that is an uniformly random element of G_T as $t \xleftarrow{\$} \mathbb{Z}_p$. \square

3.4 Construction Π'_R : More Efficient Decryption

Given a pair encoding scheme P for predicate function R , a predicate encryption Π'_R is defined as following.

- **Setup**($1^\lambda, N$): mpk and msk are same as Π_R in Sect. 3.2. Only difference is mpk now includes a one-time signature scheme OTS of its choice.
- **KeyGen**(msk, x): Same as KeyGen of Π_R in Sect. 3.2.
- **Encrypt**(mpk, y, M): Same as Encrypt of Π_R in Sect. 3.2. Only difference being it runs $(vk, sk) \leftarrow \text{OTS.Gen}(1^\lambda)$. Then it computes $\xi = \mathcal{H}(\mathbf{C}, vk)$ where \mathbf{C} is computed exactly the same as presented in Encrypt of Π_R in Sect. 3.2

and outputs $\bar{\mathbf{C}} = (\bar{\mathbf{C}}_0, \mathbf{C}, vk, \sigma)$ where $\bar{\mathbf{C}}_0 = g_1^{(\xi \mathbf{W}_{n+1} + \mathbf{W}_{n+2}) \mathbf{B} \begin{pmatrix} s_0 \\ 0 \end{pmatrix}}$ and $\sigma \leftarrow \text{OTS.Sign}(sk, \bar{\mathbf{C}}_0)$.

– **Decrypt**($\mathbf{K}, \bar{\mathbf{C}}$): It differs from the **Decrypt** of Π_R in Sect. 3.2. Given \mathbf{K} and $\bar{\mathbf{C}}$ corresponding to key-index x and data-index y respectively, if $R(x, y) = 0$, it aborts. Also aborts if $\text{OTS.Verify}(\bar{\mathbf{C}}_0, vk, \sigma)$ evaluates to 0. Otherwise runs $\mathbf{E} \leftarrow \text{Pair}(x, y, N)$. Given $\mathbf{K} = (\mathbf{K}_1, \dots, \mathbf{K}_{m_1})$ and ciphertext $\bar{\mathbf{C}}$ it computes $(\hat{\mathbf{K}}_1, \dots, \hat{\mathbf{K}}_{w_1})$ where $\hat{\mathbf{K}}_{\tilde{i}} = \prod_{\iota \in [m_1]} (\mathbf{K}_\iota)^{\mathbf{E}_{\iota \tilde{i}}}$ for each $\tilde{i} \in [w_1]$. Chooses $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_p^d$.

Defines modified key $\hat{\mathbf{K}} = (\hat{\mathbf{K}}_0, \hat{\mathbf{K}}_1, \dots, \hat{\mathbf{K}}_{w_1})$ where $\hat{\mathbf{K}}_0 = g_2^{\mathbf{z} \begin{pmatrix} \mathbf{r} \\ 0 \end{pmatrix}}$, $\hat{\mathbf{K}}_1 = \Phi \cdot \tilde{\mathbf{K}}_1$ for $\Phi = g_2^{(\xi \mathbf{W}_{n+1}^\top + \mathbf{W}_{n+2}^\top) \mathbf{z} \begin{pmatrix} \mathbf{r} \\ 0 \end{pmatrix}}$ and $\xi = \mathcal{H}(\mathbf{C}, vk)$ and $\hat{\mathbf{K}}_i = \tilde{\mathbf{K}}_i$ for $i \in [2, w_1]$. Outputs M such that

$$M = C_{w_1+1} \cdot e(\bar{\mathbf{C}}_0, \hat{\mathbf{K}}_0) \cdot \left(\prod_{\tilde{i} \in [w_1]} e(C_{\tilde{i}}, \hat{\mathbf{K}}_{\tilde{i}}) \right)^{-1}. \quad (3)$$

Correctness. See full version of this paper [19].

Remark 4. This construction is quite similar to the one presented in Sect. 3.2. However, the ciphertext now has extra two elements. Here, we compute the hash of (\mathbf{C}, vk) first and bind it to the randomness $\mathbf{B} \begin{pmatrix} s_0 \\ 0 \end{pmatrix}$ using common variables \mathbf{W}_{n+1} and \mathbf{W}_{n+2} where vk is verification key for OTS. This results in an extra ciphertext component namely $\bar{\mathbf{C}}_0$. We then use the one-time signature OTS to compute a signature σ on $\bar{\mathbf{C}}_0$ and output $\bar{\mathbf{C}} = (\bar{\mathbf{C}}_0, \mathbf{C}, vk, \sigma)$. Use of OTS ensures *integrity* of $\bar{\mathbf{C}}_0$ thereby allowing us to get rid of extra verification step (precisely Eq. (1) in Sect. 3.2) that is needed to check the structure of $\bar{\mathbf{C}}_0$.

Remark 5. Even if our construction uses OTS for CCA-security, the technique is quite different than that of CHK [18] and its descendants. All those schemes essentially depends on the key-delegation capability of the underlying CPA-secure (H)IBE. Yamada et al. [10] formalized this notion as property of *delegatability*. We note that not all pair encoding based predicate encryptions achieve this property (see Table 3). Even for schemes that achieve delegatability, one needs to apply *index transformers* [12] on both key and data index. This often makes the resultant ciphertext and secret key significantly large (see [12, Table 1] for details) thereby degrading the decryption performance.

Efficiency. Our construction increases the ciphertext length by exactly three components namely $\bar{\mathbf{C}}_0$, vk and σ will be returned along with the CPA-ciphertext \mathbf{C} where $\bar{\mathbf{C}}_0 \in G_1^{(d+1)}$, vk is verification key of OTS and σ is the signature for $\bar{\mathbf{C}}_0$ with respect to the signing key sk corresponding to vk . As we mentioned earlier, we have reused **KeyGen** of [5], therefore the secret key does not change. However **Decrypt** has to verify the signature σ and evaluate only one additional unit of pairing (namely $e(\bar{\mathbf{C}}_0, \hat{\mathbf{K}}_0)$). As both $\bar{\mathbf{C}}_0$ and $\hat{\mathbf{K}}_0$ are group elements having $(d+1)$ -components, the decryption in our construction incurs an additional cost of $(d+1)$ pairing evaluations only. This is more efficient than Π_R (of Sect. 3.2).

3.5 Security of Π'_R

Theorem 2. *Suppose a regular decryption pair encoding scheme P for predicate R is both SMH-Secure and CMH-Secure in \mathcal{G} , and the \mathcal{D}_d -Matrix DH Assumption holds in \mathcal{G} . Then the scheme Π'_R is fully CCA-secure encryption scheme if \mathcal{H} is collision resistant hash function and OTS is strong one-time signature. More precisely, for any PPT adversary \mathcal{A} that makes at most q_1 key queries before challenge, at most q_2 key queries after challenge and at most Q decryption queries throughout the game, there exists PPT algorithms $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4, \mathcal{B}_5$ such that for any λ ,*

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\Pi'_R}(\lambda) \leq & (2q_1 + 2Q + 3) \cdot \text{Adv}_{\mathcal{B}_1}^{\mathcal{D}_d\text{-MatDH}}(\lambda) + q_1 \cdot \text{Adv}_{\mathcal{B}_2}^{\text{CMH}}(\lambda) + \text{Adv}_{\mathcal{B}_3}^{\text{SMH}}(\lambda) \\ & + Q \cdot \text{Adv}_{\mathcal{B}_4}^{\text{CRH}}(\lambda) + Q \cdot \text{Adv}_{\mathcal{B}_5, \text{OTS}}^{\text{sUf-CMA}}(\lambda). \end{aligned}$$

The theorem is formally proved in the full version [19] of this paper.

3.6 Performance Comparison

In this section we provide concrete comparison, outlined in Table 1, of performance between *conventional* conversions [10–13] and our constructions over few examples. The table below compares *delegation*-based, *verifiability*-based and our (*direct*) CCA-construction technique on pair encoding based prime-order instantiation of CP-ABE, KP-ABE [20] and KP-FE for DFA [5].

For all the candidate schemes considered in the table, $vk \in \{0, 1\}^\ell$. For CP-ABE and KP-ABE, A denotes attribute set, Γ denotes the access structure such that $I \subset A$ are the attributes that satisfy Γ and *dummy attribute* set W to accommodate vk such that $|W| = 2\ell$, where Γ can be expressed as an LSSS matrix [20] of dimension $m \times k$. In case of KP-FE for DFA, $M = (Q, \Sigma, \text{Tr}, q_0, F)$ denotes

Table 3. Concrete comparison of efficiency (The “dark-gray” (resp. “light-gray”) has been used to denote *delegation* (resp. *verifiability*)-based construction while parameters and complexity of Π_R and Π'_R (i.e. *direct* constructions) are kept uncolored. Precisely, Π_R is presented at the top among the two uncolored rows for each example. Sometimes, adjacent cells have been merged if corresponding complexities are same. The two-columns under “Decryption Cost” follows the same convention as in Table 1 where the *first* cell is underlying CPA-Decryption cost and *second* cell contains *additional* cost to achieve CCA-Decryption.).

Scheme	Key	Ciphertext	Decryption Cost	
CP-ABE [20]	$2(A + 2\ell) + 2$	$2(m + \ell) + 1$	$(I + \ell)[E] + (I + \ell + 2)[P]$	$(A + \ell + 2)[E]$
	$2 A + 2$	$2(m + \ell) + 1$	$I[E] + (I + 2)[P]$	$(2I + 2\ell + 1)d[P]$
	$2 A + 2$	$2m + 1$	$I[E] + (I + 2)[P]$	$2[E] + 3[P]$ $2[E] + [P]$
KP-ABE [20]	NA	NA	NA	NA
	$2m$	$ A + \ell + 1$	$I[E] + (I + 1)[P]$	$(2I + 2\ell)d[P]$
	$2m$	$ A + 1$	$I[E] + (I + 1)[P]$	$2[E] + 3[P]$ $2[E] + [P]$
KP-FE for DFA[5]	$3(\text{Tr} + \ell) + 5$	$2(\omega + \ell) + 5$	$(3(\omega + \ell) + 5)[P]$	$\mathcal{O}((\text{Tr} + Q + \ell)(\text{Tr} + d + \ell))[E]$ $2(\omega + \ell)d[E] + (3(\omega + \ell) + 7)[P]$
	$3 \text{Tr} + 5$	$2 \omega + 5$	$(5 + 3 \omega)[P]$	$2[E] + 3[P]$ $2[E] + [P]$

the DFA and ω denotes the string. Here [E] and [P] denote number of *unit* group multiplication and number of *unit* pairing evaluations respectively. By *unit* group multiplication (resp. pairing evaluation) we mean $d + 1$ many group multiplication (resp. pairing evaluations) in a prime-order system of $d + 1$ dimension. For standard assumptions like SXDH and D-Linear, d is 1 and 2 respectively.

Note that in Table 3, we considered both KP-ABE and CP-ABE to be *small-universe*. The additional cost in CP-ABE and KP-ABE instances mentioned there, is actually the cost of performing *verifiability-1* whereas we presented the additional cost in case of KP-FE for DFAs is a sort of *public verifiability* to guarantee *verifiability-2* of the underlying CPA-decrypt algorithm (see [11, 13] for these notions of verifiability). Even if, delegation-based conversions are better than verifiability-based conversions in terms of efficiency, there are several schemes for which delegation is still unachieved. One such example is KP-ABE construction of [20] and has been marked NA (i.e. not available) in the above table. Both our direct constructions (Π_R and Π'_R) overcome these problems without affecting the performance. We emphasize that even if delegation is available for some CPA-secure CP-ABE, viz, large universe CP-ABE schemes, the delegation-based conversion is simply not applicable due to efficiency problem. For example, prime order instantiation ([5]) of large universe CP-ABE schemes for [3, Pair Encoding Scheme 13] and dual of [3, Pair Encoding Scheme 4] has transformed key-index size *exponentially big* due to introduction of *dummy attribute* $W = \{0, 1\}^\ell$.

4 Conclusion

In this work, we presented two direct adaptive CCA-secure predicate encryption constructions to convert adaptive CPA-secure predicate encryption of [5]. The ciphertext of our first construction contains *only one* additional component than in case of adaptive CPA-secure predicate encryption of [5] and decryption needs *exactly three unit* (i.e. $3 \times (d + 1)$) additional pairing evaluations. The ciphertext of our second construction, on the other hand, contains *exactly three* additional components (a $G_1^{(d+1)}$ element, an OTS verification key and a signature) than in case of adaptive CPA-secure predicate encryption of [5] and decryption needs *only one unit* additional pairing evaluations. This is a significant improvement over the previous generic conversion mechanisms which needed almost double of $m_1 \times w_1 \times (d + 1) \times (m_2 + 1) \times d$ many pairing evaluations. A possible future work might be instantiation of our generic CPA-to-CCA conversion on the predicate encryption resulted from integration of *dual system groups* with *pair encoding schemes*.

References

1. 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
2. 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
3. Attrapadung, N.: Dual system encryption via doubly selective security: framework, fully secure functional encryption for regular languages, and more. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 557–577. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_31
4. Wee, H.: Dual system encryption via predicate encodings. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 616–637. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_26
5. Attrapadung, N.: Dual system encryption framework in prime-order groups via computational pair encodings. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 591–623. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_20
6. 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
7. Chen, J., Wee, H.: Dual system groups and its applications – compact HIBE and more. Cryptology ePrint Archive, Report 2014/265 (2014). <http://eprint.iacr.org/2014/265>
8. Agrawal, S., Chase, M.: A study of pair encodings: predicate encryption in prime order groups. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 259–288. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_10
9. Agrawal, S., Chase, M.: Simplifying design and analysis of complex predicate encryption schemes. Cryptology ePrint Archive, Report 2017/233 (2017). <http://eprint.iacr.org/2017/233>
10. Yamada, S., Attrapadung, N., Hanaoka, G., Kunihiro, N.: Generic constructions for chosen-ciphertext secure attribute based encryption. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 71–89. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19379-8_5
11. Yamada, S., Attrapadung, N., Santoso, B., Schuldt, J.C.N., Hanaoka, G., Kunihiro, N.: Verifiable predicate encryption and applications to CCA security and anonymous predicate authentication. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 243–261. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_15
12. Nandi, M., Pandit, T.: Generic conversions from CPA to CCA secure functional encryption. Cryptology ePrint Archive, Report 2015/457 (2015). <http://eprint.iacr.org/>
13. Nandi, M., Pandit, T.: Verifiability-based conversion from CPA to CCA-secure predicate encryption. J. Appl. Algebra Eng. Commun. Comput., 1–26 (2017). <https://doi.org/10.1007/s00200-017-0330-2>
14. Blömer, J., Liske, G.: Construction of fully CCA-secure predicate encryptions from pair encoding schemes. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 431–447. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_25

15. Nandi, M., Pandit, T.: On the power of pair encodings: Frameworks for predicate cryptographic primitives. Cryptology ePrint Archive, Report 2015/955 (2015). <http://eprint.iacr.org/2015/955>
16. Freeman, D.M.: Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 44–61. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_3
17. Boyen, X., Mei, Q., Waters, B.: Direct chosen ciphertext security from identity-based techniques. In: ACM Conference on Computer and Communications Security, CCS 2005, pp. 320–329. ACM (2005). <https://doi.org/10.1145/1102120.1102162>
18. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_13
19. Chatterjee, S., Mukherjee, S., Pandit, T.: CCA-secure predicate encryption from pair encoding in prime order groups: generic and efficient. Cryptology ePrint Archive, Report 2017/657 (2017). <http://eprint.iacr.org/2017/657>
20. 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

Cold Boot Attacks on NTRU

Kenneth G. Paterson^(✉) and Ricardo Villanueva-Polanco

Information Security Group, Royal Holloway, University of London, Egham, UK
{kenny.paterson,ricardo.villanuevapolanco.2013}@rhul.ac.uk

Abstract. Cold boot attacks target memory remanence effects in hardware to secret key material. Such attacks were first explored in the scientific literature by Halderman et al. (USENIX Security Symposium 2008) and, since then, different attacks have been developed against a range of asymmetric key and symmetric key algorithms. Such attacks in general receive as input a noisy version of the secret key as stored in memory, and use redundancy in the key (and possibly knowledge of a public key) to recover the secret key. The challenge is to recover the key as efficiently as possible in the face of increasing levels of noise. For the first time, we explore the vulnerability of lattice-based cryptosystems to this form of analysis, focussing in particular on NTRU, a well-established and attractive public-key encryption scheme that seems likely to be a strong candidate for standardisation in NIST’s post-quantum process. We look at two distinct NTRU implementations, showing how the attacks that can be developed depend critically on the in-memory representation of the secret key. We develop, efficient, dedicated key-recovery algorithms for the two implementations and provide the results of an empirical evaluation of our algorithms.

Keywords: Cold boot attacks · NTRU · Key enumeration

1 Introduction

Cold boot attacks have received significant attention since they were first described in the literature by Halderman et al. nearly a decade ago [7] (see also [8]). This class of attack relies on the fact that computer memory normally keeps information when going through a power-down/power-up cycle, so an adversary might be able to gain access to confidential information such as cryptographic keys after a system reboot. Unluckily for such an adversary, once the power is cut off, the bits in memory will undergo a gradual degradation, meaning that any information retrieved from the computer memory will probably be noisy. Thus, once the location of the key in memory has been discovered (itself a non-trivial task), the adversary’s task becomes the mathematical problem of recovering a key from a noisy version of that key. The adversary may have access to reference cryptographic data created using that key (e.g. ciphertexts for a symmetric key encryption scheme) or have a public key available (in the asymmetric setting).

The amount of time for which information is maintained while the power is off depends on the particular memory type and the ambient temperature. Experimental results shown in [7] reveal that, at normal operating temperatures, there is little corruption within the first few seconds, but this phase is then followed by a rapid decay. The period of mild corruption can be prolonged by cooling the memory chips. For instance, according to [7], in an experiment at -50°C (which can be achieved by spraying compressed air onto the memory chips) less than 0.1% of bits decay within the first minute. At temperatures of approximately -196°C (achieved by means of the use of liquid nitrogen) less than 0.17% of bits decay within the first hour. Notably, once power has been switched off, the memory will be partitioned into regions, and each region will have a 'ground state' which is associated with a bit, 0 or 1. In a 0 ground state, the 1 bits will eventually decay to 0 bits, while the probability of a 0 bit switching to a 1 bit is very small, but not vanishing (a common probability is circa 0.001 [7]). When the ground state is 1, the opposite is true. An attacker can determine the ground state of a particular region of memory rather easily in an attack by reading all the bits and determining how many of them are 0 bits and how many are 1 bits.

The main focus of cold boot attacks after the initial work pointing out their feasibility [7] has been to develop algorithms for efficiently recovering keys from noisy versions of those keys for a range of different cryptosystems, whilst exploring the limits of how much noise can be tolerated. Heninger and Shacham [10] focussed on the case of RSA keys, giving an efficient algorithm based on Hensel lifting to exploit redundancy in the typical RSA private key format. This work was followed up by Henecka et al. [9] and Paterson et al. [18], with both papers also focussing on the mathematically highly structured RSA setting. The latter paper in particular pointed out the *asymmetric* nature of the error channel intrinsic to the cold boot setting and recast the problem of key recovery for cold boot attacks in an information theoretic manner. Cold boot attacks in the discrete logarithm setting were considered in [19]. There, the authors emphasise the critical role of the format in which the private key is stored in memory in the development and success of attacks. Several papers have considered cold boot attacks in the symmetric key setting, including Albrecht and Cid [1] who focussed on the recovery of symmetric encryption keys in the cold boot setting by employing polynomial system solvers, and Kamal and Youssef [14] who applied SAT solvers to the same problem. Further research on the development of cold boot attacks for specific schemes can be found in [13, 15]. Cold boot attacks are also widely cited in the theoretically-oriented literature on leakage-resilient cryptography, but the relevance there is marginal because the cold boot attack scenario (direct access to a noisy version of the whole key) does not really apply in the leakage-resilient setting.

1.1 Our Contributions

In this paper, we examine the feasibility of cold boot attacks against the NTRU public key encryption scheme [11, 12]. We believe this to be the first time that

this has been attempted. Our work can be seen as a continuation of the trend to develop cold boot attacks for different schemes (as evinced by the literature cited above). But it can also be seen as the beginning of the evaluation of the leading post-quantum candidates against this class of attack. Such an evaluation should form a small but important part of the overall assessment of schemes in the soon-to-commence NIST selection process for post-quantum algorithms.¹ In particular, this paper evaluates what seems likely to be a leading candidate and lays the groundwork for the later study of other likely candidates in the same broad family of schemes that operate over polynomial rings (such as NTRUprime [3] and various recently proposed ring-LWE-based schemes [2, 5]).

As noted above, the exact format in which the private key is stored is critical to developing key recovery attacks in the cold boot setting. This is because the attack depends on physical effects in memory, represented by bit flips in private key bits, and the main input to the attack is a bit-flipped version of the private key. For this reason, it is necessary to either propose natural ways in which keys would be stored in memory in NTRU implementations or to examine specific implementations of NTRU. We adopt the latter approach, and we study two distinct implementations. The first, `ntru-crypto`, is a pair of C and Java libraries developed by OnBoard Security, a spin off of Security Innovation, the patent-holder for some NTRU technology.² The second, `tbuktu` is a pair of libraries developed by “Tim Buktu”, and is available in ‘C’ and Java languages.³ A fork of the Java implementation is included in the popular Bouncy Castle Java crypto library.⁴

Each of these implementations stores its private keys in memory in slightly different ways. For example, in Java, `tbuktu` supports a number of different formats, including a representation where the key is stored as 6 lists of indices, each index being a 32-bit integer representing a position where a certain polynomial has a coefficient of value +1 or -1. Meanwhile, `ntru-crypto`’s C implementation uses a special representation of polynomial coefficients by trits (three-valued bits), and then packs 5 trits at a time into octets using base-3 arithmetic.

Each of these different private key formats therefore requires a different approach to key recovery in the cold boot setting. In this paper, we will focus on just a couple of the more interesting cases, where there is some additional structure that we can exploit, or where novel approaches are called for. Nevertheless, we will pose the problem of key recovery in a more general way that makes it possible to see how to generalise our ideas to cover other cases. Specifically, each of our analyses involves splitting the (noisy) private key into chunks, and creating log-likelihood estimates for each candidate value for each of the chunks.

¹ See <http://csrc.nist.gov/groups/ST/post-quantum-crypto/> for details of the NIST process.

² See <https://github.com/NTRUOpenSourceProject/ntru-crypto> for the code and <https://www.onboardsecurity.com/products/ntru-crypto/ntru-resources> for a list of useful resources related to NTRU.

³ See <http://tbuktu.github.io/ntru/>.

⁴ See <http://bouncycastle.org/>.

Each such estimate can be regarded as a per chunk score. A log-likelihood estimate (or score) for a candidate for the complete private key can then be computed by summing the per chunk scores across the different chunks. Our problem then becomes one of efficiently enumerating complete candidates and their scores based on lists of candidates for chunks and per-chunk scores, so that each complete candidate can then be tested for correctness (for example, by trial encryption and decryption). It makes sense to perform the enumeration in decreasing order of score if possible, starting with the most likely candidate. This is a problem that also arises in the side-channel attack literature, cf. [4, 6, 16, 17, 20], where, for example, one might obtain scoring information for each byte of an AES key from a power analysis attack and then want to efficiently enumerate and test a large number of complete 16-byte candidates in decreasing order of score until the correct key is found. We are able to apply standard algorithms (e.g. depth-first search on a tree with pruning) as well as algorithms from this literature to solve the key recovery problem in our context.

1.2 Paper Organisation

This paper is organised as follows. Section 2 describes cold boot attacks and the adversary model in more detail. It also gives a basic statistical approach to recovering private keys in the face of noise, based on maximum likelihood estimation. Section 3 describes the NTRU public key encryption algorithm and details of the two implementations that we target. Section 4 describes our algorithms for attacking these implementations; these are largely based on established key enumeration techniques from the literature on side-channel attacks. Section 5 describes our implementation of the algorithms and the results of our empirical evaluation of the performance of the attacks. We conclude in Sect. 6

2 Further Background

2.1 Cold Boot Attack Model

Our cold boot attack model assumes that the adversary can obtain a noisy version of the original NTRU private key (using whatever format is used to store it in memory). We assume that the corresponding NTRU public key is known exactly (without noise). We do not consider here the important problem of how to locate the appropriate area of memory in which the private key bits are stored, though this would be an important consideration in practical attacks. Our aim is then recover the private key. Note that it is sufficient to recover a list of key candidates in which the true private key is located, since we can always test a candidate by doing a trial encryption using the known public key and then decryption using the candidate. It is highly likely that a simple test of this type will filter out all wrong candidates (especially when the NTRU variant considered is CCA secure).

We assume throughout that a 0 bit of the original private key will flip to a 1 with probability $\alpha = P(0 \rightarrow 1)$ and that a 1 bit of the original private key

will flip with probability $\beta = P(1 \rightarrow 0)$. We do not assume that $\alpha = \beta$; indeed, in practice, one of these values may be very small (e.g. 0.001) and relatively stable over time, while the other increases over time. Furthermore, we assume that the attacker knows the values of α and β and that they are fixed across the region of memory in which the private key is located. These assumptions are reasonable in practice: one can estimate the error probabilities by looking at a region where the memory stores known values (e.g. where the public key is located), and the regions are typically large. Moreover, our algorithms will be fairly robust to mis-estimations of these parameters.

2.2 Log Likelihood Statistic for Key Candidates

Suppose we have a true private key that is W bits in size, and let $\mathbf{r} = (r_0, \dots, r_{W-1})$ denote the bits of the noisy key (input to the adversary in the attack). Suppose a key recovery algorithm constructs a candidate for the private key $\mathbf{c} = (c_0, \dots, c_{W-1})$ by some means (to be determined). Then, given the bit-flip probabilities α, β , we can assign a likelihood score to \mathbf{c} as follows:

$$L[\mathbf{c}; \mathbf{r}] := \Pr[\mathbf{r}|\mathbf{c}] = (1 - \alpha)^{n_{00}} \alpha^{n_{01}} \beta^{n_{10}} (1 - \beta)^{n_{11}}$$

where n_{00} denotes the number of positions where both \mathbf{c} and \mathbf{r} contain a 0 bit, n_{01} denotes the number of positions where \mathbf{c} contains a 0 bit and \mathbf{r} contains a 1 bit, etc.

The method of maximum likelihood estimation⁵ then suggests picking as \mathbf{c} the value that maximises the above expression. It is more convenient to work with log likelihoods, and equivalently to maximise these, viz:

$$\mathcal{L}[\mathbf{c}; \mathbf{r}] := \log \Pr[\mathbf{r}|\mathbf{c}] = n_{00} \log(1 - \alpha) + n_{01} \log \alpha + n_{10} \log \beta + n_{11} \log(1 - \beta).$$

We will frequently refer to this log likelihood expression as a *score* and seek to maximise its value (or, equally well, minimise its negative).

2.3 Combining Chunks to Build Key Candidates

Now suppose that the true private key \mathbf{r} can be represented as a concatenation of W/w chunks, each on w bits. As we shall see in the specific analyses of different NTRU implementations, this will be the case in practice. For example, each chunk might arise from the value of a coefficient of some polynomial making up the NTRU private key.

Let us name the chunks $r^0, r^1, \dots, r^{W/w-1}$ so that $r^0 = r_0 r_1 \dots r_{w-1}$, $r^1 = r_w r_{w+1} \dots r_{2w-1}$, etc. Suppose also that candidates \mathbf{c} can be represented by concatenations of chunks $c^0, c^1, \dots, c^{W/w-1}$ in the same way.

Suppose further that each of the at most 2^w candidates for chunk c^i ($0 \leq i < W/w$) can be enumerated and given its own score by some procedure (formally, a sub-algorithm in an overall attack). For example, the above expression for log

⁵ See for example https://en.wikipedia.org/wiki/Maximum_likelihood_estimation.

likelihood across all W bits of private key is easily modified to produce a log likelihood expression for any candidate for chunk i as follows:

$$\mathcal{L}[c^i; r^i] := \log \Pr[r^i | c^i] = n_{00}^i \log(1 - \alpha) + n_{01}^i \log \alpha + n_{10}^i \log \beta + n_{11}^i \log(1 - \beta) \tag{1}$$

where the n_{ab}^i values count occurrences of bits across the i -th chunks, r^i, c^i .

Thus we can assume that we have access to W/w lists of scores, each list containing up to 2^w entries. Note that W/w scores, one from each of these per-chunk lists, can be added together to create a total score for a complete candidate \mathbf{c} . Indeed, this total score is statistically meaningful in the case where the per-chunk scores are log likelihoods because of the additive nature of the scoring function in that case.

The question then becomes: can we devise efficient algorithms that traverse the lists of scores to combine chunk candidates c^i , obtaining complete key candidates \mathbf{c} having high total scores (with total scores obtained by summation)? As noted in the introduction, this is a problem that has been previously addressed in the side-channel analysis literature [4, 6, 16, 17, 20], with a variety of different algorithmic approaches being possible to solving the problem. We shall return to this question after having described the specific NTRU implementations that we will attack.

3 NTRU Encryption Scheme and Private Key Formats

In this section we briefly describe the NTRU public key encryption scheme and explore the various private key formats in the two implementations we will be working with.

Let $N, p, q \in \mathbb{Z}^+$. We define three polynomial rings:

$$R = \mathbb{Z}[x]/(X^N - 1), \quad R_p = \mathbb{Z}_p[x]/(X^N - 1), \quad R_q = \mathbb{Z}_q[x]/(X^N - 1).$$

Thus, for example, elements of R_p can be represented as polynomials of degree at most $N - 1$ with coefficients from \mathbb{Z}_p . They can also be represented as vectors of dimension N over \mathbb{Z}_p in the natural way, and we will switch between representations at will.

Definition 1. Let $\mathbf{a} \in R_q$. The centred lift of \mathbf{a} to R is the unique polynomial $\mathbf{a}' \in R$ satisfying $\mathbf{a}' \bmod q = \mathbf{a}$ whose coefficients all lie in the interval $-\frac{q}{2} < a'_i \leq \frac{q}{2}$.

Definition 2. Let r be a fixed integer and let C_r be the function that, given $\mathbf{a} \in R$, outputs the number of coefficients of \mathbf{a} equal to r . Let $d_1, d_2 \in \mathbb{Z}^+$. We define $T(d_1, d_2) = \{\mathbf{a} \in R \mid C_1(\mathbf{a}) = d_1, C_{-1}(\mathbf{a}) = d_2, C_0(\mathbf{a}) = N - d_1 - d_2\}$. Note that $|T(d_1, d_2)| = \binom{N}{d_1} \binom{N-d_1}{d_2}$. An element $\mathbf{a} \in R$ is called a ternary polynomial if and only if $\mathbf{a} \in T(d_1, d_2)$ for some $d_1, d_2 \in \mathbb{Z}^+$.

3.1 NTRU Public Key Encryption Scheme

The NTRU public key encryption scheme is a lattice-based alternative to RSA and ECC with security that is (informally) based on the problem of finding the shortest vector in a particular class of lattices. The scheme exists in several different versions, offering different forms of security (IND-CPA, IND-CCA). The details of the scheme's operation matter less to us than the format of private keys in implementations. However, for completeness, we give an overview of NTRU. We follow the description in [11].

The scheme relies on public parameters (N, p, q, d) with N and p prime, $\gcd(p, q) = \gcd(N, q) = 1$ and $q > (6d + 1)p$.

Key generation:

1. Choose $\mathbf{f} \in T(d + 1, d)$ that is invertible in R_q and R_p .
2. Choose $\mathbf{g} \in T(d_1, d_2)$ for some $d_1, d_2 \in \mathbb{Z}^+$.
3. Compute \mathbf{f}_p , the inverse of \mathbf{f} in R_p .
4. Compute \mathbf{f}_q , the inverse of \mathbf{f} in R_q .
5. The public key is $\mathbf{h} = p\mathbf{f}_q \cdot \mathbf{g} \in R_q$; the private key is the pair $(\mathbf{f}, \mathbf{f}_p)$.

Encryption: On input message \mathbf{m} , which we assume to be a centre-lifted version of an element of R_p , and public key \mathbf{h} :

1. Choose a random \mathbf{r} with small coefficients, in particular \mathbf{r} can be chosen such that $\mathbf{r} \in T(d, d)$.
2. Compute the ciphertext \mathbf{e} as $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m} \in R_q$.

Decryption: On input ciphertext \mathbf{e} and private key $(\mathbf{f}, \mathbf{f}_p)$:

1. Compute $\mathbf{b} = \mathbf{f} \cdot \mathbf{e}$ in R_q . (Note that this yields $\mathbf{b} = p\mathbf{g} \cdot \mathbf{r} + \mathbf{f} \cdot \mathbf{m}$ over R_q .)
2. Centre-lift \mathbf{b} modulo q to obtain \mathbf{a} , and then compute $\mathbf{f}_p \cdot \mathbf{a} \in R_p$. Centre-lift the result modulo p to obtain \mathbf{m}' .

We omit the correctness proof for this description of the NTRU scheme. Note that \mathbf{f}_p can be computed from \mathbf{f} on the fly, and so some implementations may only store \mathbf{f} as the private key.

3.2 Private Key Formats for NTRU Implementations

NTRU at first was only available as a proprietary, paid-for library. It was not until 2011 that the first open-source implementation, `tbuktu`, appeared under a BSD licence.⁶ A fork of this first implementation forms the basis of the NTRU code in the Bouncy Castle library. Two years later Security Innovation exempted open source projects from having to obtain a patent license for their `ntru-crypto` implementation⁷ and released an NTRU reference implementation under the GPL v2 licence. Each of these two implementations is available in both Java and C. We examine the private key formats for each of these implementations in turn.

⁶ See <http://tbuktu.github.io/ntru/>.

⁷ See <https://github.com/NTRUOpenSourceProject/ntru-crypto/blob/master/FOSS%20Exception.md>.

3.2.1 The `tbuktu`/Bouncy Castle Java Implementation

In this implementation, there are four pieces of information that determine how the private key is stored:

1. A variable `t` that points to a polynomial and from which variables corresponding to the private key components `f` and `fp` are constructed.
2. A variable `polyType` that indicates the type of polynomial to use. This can hold two values: `SIMPLE` or `PRODUCT`.
3. A boolean `sparse` that indicates if `t` is an sparse or dense polynomial. This variable applies only if `polyType` has value `SIMPLE`.
4. A boolean `fastFp` that indicates the manner in which `f` is built from `t`. If `fastFp = true`, then $p = 3$, $f = 1 + 3t$ and $f_p = 1$; otherwise $f = t$ and $f_p = t^{-1} \bmod p$. This relates to an implementation trick for the case $p = 3$.

When `polyType` has the value `SIMPLE`, `t` will be either a dense ternary polynomial or a sparse ternary polynomial, as determined by the value of `sparse`. In the dense case, `t` is represented as an `int` array of length N whose entries have values from $\{-1, 0, 1\}$. In memory, each entry is stored as a 32-bit signed integer, using two's complement, i.e., $+1$ is stored as the 32-bit string `000...01`, 0 is stored as `000...00` and -1 is stored as `111...11`. Meanwhile, in the sparse case, `t` is represented as two `int` arrays, `ones` and `negOnes`, where:

1. The array `ones` contains the indices of the $+1$ coefficients of `t` in increasing order (so that the entries in the array are 32-bit representations of integers in the range $[0, N - 1]$).
2. The array `negOnes` contains the indices of the -1 coefficients of `t` in increasing order (with entries having the same bit representation as the entries of `ones`).

When `polyType` has the value `PRODUCT`, `t` will be a product form polynomial. In this case, `t` is represented by three different sparse ternary polynomials `f1`, `f2`, `f3` such that $t = f_1 f_2 + f_3$. All three of `f1`, `f2`, `f3` are stored in memory separately in sparse form. This means that, when `polyType` has the value `PRODUCT`, then the private key is represented in memory by a total of 6 `int` arrays `fi.ones`, `fi.negOnes`, $1 \leq i \leq 3$.

Note that the private key formats for the `tbuktu` C implementation are largely the same as for the Java one, and so we do not detail them further here.

3.2.2 Reference Parameters for `tbuktu`

The `tbuktu` implementation includes 10 named reference parameter sets with a range of choices for N and q , targeting different security levels and optimisations. These 10 sets are detailed in the `EncryptionParameters` class.

For example, both `APR2011_439` and `APR2011_439_FAST` parameter sets target 128 bits of security. If the former set is selected, $f = t$, with `t` being represented as a sparse ternary polynomial with `df` = 146 coefficients set to $+1$ and with `df` - 1 of them set to -1 . If the latter set is selected, then $f = 1 + 3t$, with $t = f_1 f_2 + f_3$; moreover `f1` has `df1` = 9 coefficients set to each of $+1$ and -1 , so $f_1 \in T(9, 9)$ (while `df2` = 8 and `df3` = 5 for `f2` and `f3`, respectively).

3.2.3 The ntru-crypto Java Implementation

Here, the private key \mathbf{f} is always of the form $1 + 3\mathbf{t}$ where \mathbf{t} is a ternary polynomial and we have $p = 3$ (so that $\mathbf{f}_p = 1$). In this implementation, \mathbf{f} is stored directly in memory as an array of short integers. That is, the coefficients f_0, f_1, \dots, f_{N-1} of \mathbf{f} are stored as a sequence of 16-bit signed two's complement integers with $f_0 \in \{-2, 1, 4\}$ and $f_i \in \{-3, 0, 3\}$, for $1 \leq i < N$.

3.2.4 The ntru-crypto C Implementation

Here, key generation is carried out by the function `ntru_crypto_ntru_encrypt_keygen`. During its execution, \mathbf{f} (the private key) is initially generated as either a product of polynomials ($\mathbf{f} = \mathbf{f}_1 \cdot \mathbf{f}_2 + \mathbf{f}_3$) or as a single polynomial. Either way, \mathbf{f} is represented internally as a list of the indices of the $+1$ coefficients followed by a list of the indices of the -1 coefficients, where each index is stored in an unsigned 16-bit integer. This data is then used to construct a packed private key blob following one of two formats. The information-dense nature of these formats makes it harder to mount cold boot key recovery attacks that perform significantly better than a combinatorial search based on searching over low-weight error patterns. For this reason we do not consider this format any further in this paper.

3.2.5 Reference Parameters for ntru-crypto

The `ntru-crypto` implementation includes 12 named reference parameter sets with a range of choices for N and q , targeting different security levels and optimisations. For the Java implementation, these parameter sets are defined in the `KeyParams` class. The values of N range from 401 to 1499, with $p = 3$ and $q = 2048$ throughout; the number of $+1$'s and -1 's in \mathbf{f} (resp. \mathbf{g}), denoted \mathbf{df} (resp. \mathbf{dg}) depends on N ; for example, for the parameter set `ees449ep1`, we have $N = 449$, $\mathbf{df} = 134$, and $\mathbf{dg} = 149$.

4 Mounting Cold Boot Key Recovery Attacks

In this section, we present our cold boot key recovery attacks on the implementations and corresponding private key formats introduced in the previous section. First, because of its simplicity, we consider the `ntru-crypto` Java Implementation (in which \mathbf{f} is stored directly in memory as an array of short integers). We will then consider the `tbuktu` Java Implementation in which the `PRODUCT` form of private key is used and such that `fastFp = true`, so that $\mathbf{f} = 1 + 3\mathbf{t}$ with $\mathbf{t} = \mathbf{f}_1\mathbf{f}_2 + \mathbf{f}_3$ where all three of $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ are stored in memory in sparse form.

We continue to make the assumptions outlined in Sect. 2. We additionally assume that all relevant public parameters and private key formatting information are known to the adversary.

4.1 The ntru-crypto Java Implementation

Recall from Sect. 3.2.3 that the coefficients of \mathbf{f} are stored directly in memory as an array of 16-bit, signed two’s complement integers, with $f_0 \in \{-2, 1, 4\}$ and $f_i \in \{-3, 0, 3\}$, for $1 \leq i < N$. For simplicity, we assume that f_0 is known (there are only 3 possible values for f_0 and the attack can be repeated for each possible value). The attacker then receives a noisy version $\mathbf{r} = (r_0, \dots, r_{W-1})$ of the array with entries f_1, \dots, f_{N-1} which is $W = 16(N-1)$ bits in size. In the terminology of Sect. 2, we set $w = 16$ and partition the noisy key into $W/w = N-1$ chunks r^i , each chunk corresponding to a single, 16-bit encoded coefficient f_{i+1} .

Using Eq. (1), we can compute log-likelihood scores $\mathcal{L}[c^i; r^i]$ for each chunk i and each candidate c^i for that chunk. Note that in each chunk, there are only 3 possible candidates c^i , since $f_i \in \{-3, 0, 3\}$. (In the general formulation with $w = 16$ there could be up to 2^{16} candidates per chunk.)

Hence we obtain $N-1$ lists of scores (log-likelihood values), each list containing 3 values. Alternatively, we can think of this as being an array of size $3 \times (N-1)$. Our task now is to combine candidates, one per chunk, to generate complete private key candidates \mathbf{c} with high log-likelihoods, which can then be tested via trial encryption and decryption.

In order to generate complete private key candidates \mathbf{c} with high scores, we employ an algorithm that is closely based on that of [17] from the side-channel attack literature. Specifically, as in [17], we use a standard depth-first search across the chunk counter i to enumerate candidates. This employs a stack, with partial cumulative scores for candidates at “depth” i in the search being computed by adding the chunk score at depth i to a cumulative score for the candidates at “depth” $i-1$. Once “depth” $N-1$ is reached, and a complete candidate is generated, the candidate can be filtered and then tested. (In fact, the known restrictions on the number of +3 and -3 coefficients in private keys for the standard parameters that we are attacking can be used to perform early aborts on partial candidates.)

As in [17], we can restrict the search space to certain intervals of scores by appropriate pruning of partial solutions. By representing a complete search space as a union of intervals, a degree of parallelisation can be achieved (but this may involve repeated computation). We can also adapt the approaches of [16, 17] to perform enumeration rather than generating candidates – computing *how many* candidates have scores in a given interval. This is useful for estimating the likely performance of the search algorithm. However, a significant difference with [16, 17] arises from the parameters involved – there, typically there are 16 chunks with 256 candidates per chunk (corresponding to AES key bytes), whereas here we will have on the order of a few hundred chunks and only 3 candidates per chunk.

4.2 The tbuktu Java Implementation

Now we turn our attention to the `tbuktu` Java implementation in some of its more interesting cases. Recall from Sect. 3.2.1 that when the `PRODUCT` form of

private key is used and when `fastFp = true`, then we have $\mathbf{f} = 1 + 3\mathbf{t}$ with $\mathbf{t} = \mathbf{f}_1\mathbf{f}_2 + \mathbf{f}_3$ where all three of $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ are stored in memory in sparse form.

This means that we have 6 arrays of indices in memory $\mathbf{f}_i.\text{ones}, \mathbf{f}_i.\text{negOnes}$, $1 \leq i \leq 3$. Each array is of type `int` and each entry in each array stores the position of either a +1 or a -1 coefficient in one of the polynomials \mathbf{f}_i ; moreover the entries should be in increasing order. We assume the starting positions in memory, total sizes, and ranges of possible values in each of these tables is known. We also know that for any pair $\mathbf{f}_i.\text{ones}, \mathbf{f}_i.\text{negOnes}$, the two tables of values should be non-intersecting. We let L_i denote the common length of the two arrays $\mathbf{f}_i.\text{ones}, \mathbf{f}_i.\text{negOnes}$ (this is determined by the parameters used to generate the private key).

We now present a two-phase attack to generate complete private key candidates.

4.2.1 Phase 1

In the first phase, we apply a modified version of the *Optimal Key Enumeration Algorithm (OKEA)* of [20]. As in the description in Sect. 2, this algorithm takes as input a collection of W/w lists of candidates, one list per chunk, and produces as output a list of `lsize` complete candidates, each across all W bits. It uses a dynamic programming version of a list merging strategy to generate complete candidates in decreasing order of score. The OKEA algorithm has the property that it is guaranteed to output the `lsize` highest scoring (i.e. most likely) candidates across all the chunks (hence its optimality). It seems to be particularly effective when W/w , the number of chunks being considered, is moderate – [20] applied it in the case of reconstructing 16-byte AES keys from their bytes, with 16 chunks.

We perform this step for each of our 6 arrays as follows: we build W/w lists of candidates, setting $w = 32$ and $W = wL_i$ so that we have L_i chunks. Each chunk corresponds to one `int` entry in the array, and each list is of size N (since, at the outset, every chunk could take on any value between 0 and $N - 1$, these being the possible indices of a +1 or -1 coefficient). The score for each entry in each list is obtained using our per-chunk log-likelihood expression (1). We modify the OKEA algorithm in such a way that it is guaranteed to output the top `lsize` candidates by score which additionally respect our ordering requirement – that is, the entries in a candidate should be in increasing order of size. This modification is done by adding an extra filtering step in each merge phase of OKEA which removes candidates that do not respect the ordering constraint.

At the end of this step, then, we obtain 6 lists \mathcal{C}_i , $1 \leq i \leq 6$, the entries of each list comprising `lsize` high-scoring candidates for one of the 6 arrays $\mathbf{f}_i.\text{ones}, \mathbf{f}_i.\text{negOnes}$, $1 \leq i \leq 3$.

4.2.2 Phase 2

In the second phase of the attack, we present these 6 lists as inputs to the algorithm described in Sect. 4.1 above – that is, we perform a stack-based, depth-first search on the lists, regarding each list as giving a set of candidates on one

of 6 chunks. Each complete candidate (on 6 chunks) now gives a candidate for $\mathbf{f}_i.\mathbf{ones}, \mathbf{f}_i.\mathbf{negOnes}$, $1 \leq i \leq 3$, these being tables of the indices where the component polynomials $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ have coefficients $+1$ and -1 . We then apply the constraint that the pairs of tables be non-intersecting (applying it earlier in the process is not very efficient, since the probability of a collision of indices is small for the parameters of interest). If a candidate survives this filter, we can construct the full private key $\mathbf{f} = 1 + 3\mathbf{t}$ with $\mathbf{t} = \mathbf{f}_1\mathbf{f}_2 + \mathbf{f}_3$ and test it for correctness.

As before, this second phase is amenable to parallelisation and to searching over restricted score intervals. Now the parameters are more akin to those studied in the prior work [16, 17] – we have 6 chunks, and `lsize` candidates per chunk, with typical values for `lsize` in our experiments being 256, 512 and 1024.

5 Experimental Evaluation

5.1 Implementation

All of the algorithms discussed in this paper were implemented in Java. We choose Java for several reasons. First, the two implementations that we have studied in this paper were written in Java (as well as C). Second, the Java platform provides the Java Collections Framework to handle data structures, which reduces programming effort, and increases program speed and quality. Finally, the Java platform also easily supports concurrent programming, providing high-level concurrency APIs.

5.1.1 Parallelisation

We made extensive use of parallelisation in our implementations, particularly for the stack-based, depth-first search that is at the core of both attacks. The first parallelisation method we used comes directly from [17] and involves splitting up the range of scores of interest into n disjoint, equal-sized sub-intervals. The second method involves splitting the list of candidates for the first chunk in our algorithm into m equal-sized sub-lists, and running the algorithm as a separate task for each sub-list, thereby constraining solutions from each task to begin with a chunk from the specified sublist for that task. These two approaches can be combined, to execute mn threads in parallel. Of course, as soon as one of the threads completes and successfully finds the private key, the others can all be aborted.

5.1.2 Search Intervals

Defining appropriate search intervals on which to run our algorithms is important in guaranteeing the success of our attacks within a reasonable amount of running time. Recall that, given a collection of lists as input, each list containing candidate for chunks and their scores, our algorithms will consider all possible candidates with total scores in any specified interval $[a, b]$. We considered two distinct classes of search interval:

1. Class I intervals are the form $[\mu - W, \mu + W]$, where μ is the average score of the correct key and W is some real number that is tuned to the maximum running time available. Here μ can be computed empirically by generating many private keys, flipping their bits according to the error probabilities α, β , and then using the usual log-likelihood scoring function. Using such intervals capture the intuition that it might be better to examine key candidates that are situated around the average score, since these are more likely to be correct. This of course violates the principle of the maximum likelihood approach.
2. Class II intervals are of the form $[\max - W, \max]$, where \max is the maximum possible score and W is again a real number that can be tuned. Here, the value of \max is easily calculated by summing across the highest scoring entries in each list. Searching in such intervals better matches the approach of maximum likelihood estimation.

5.1.3 Simulations

To simulate the performance of our algorithms, we generate a private key (according to some chosen format), flip its bits according to the error probabilities α, β , and then run our chosen algorithm with selected parallelisation parameters m, n and interval definition $[a, b]$. We refer to such a run attempting to recover a single private key as a *simulation*.

For our experiments, we ran our simulations on a machine with Intel Xeon CPU E5-2667 v2 cores running at 3.30 GHz; we used up to 16 cores. In order to run our simulations concurrently, a pool of threads is initialised with a maximum number of threads given as a parameter. When a simulation is to be run and tested, it generates its various tasks according to the given parameters, each of which then is submitted to the main pool in order. After it has finished, a thread outputs either the recovered private key or null value (indicating failure to find the key) along with some statistics. Note that having a pool created with a defined number of threads helps to avoid exhausting and reusing computational resources, in contrast to creating a new thread per task.

5.2 Results for the ntru-crypto Java Implementation

Here, we only considered Class II intervals, i.e. intervals of the form $[\max - W, \max]$. To calculate suitable values for W , we used random sampling from the set of possible candidates (by choosing chunks at random from each list) in order to estimate σ , the standard deviation of the candidate scores. We then set W as $r\sigma$ and experimented with different values of r , the idea being that larger values of r would correspond to bigger intervals, including more candidates and giving a higher chance of success at the cost of more computation. We used 2^{20} candidates in sampling to estimate σ .

After manual tuning, the number of tasks was set to 3, r was set to 0.01 and the number of subintervals m was set to 1. Hence in our experiments, searches were conducted over the interval $[\max - 0.01\sigma, \max]$ with 3 tasks.

Figure 1a shows the success rate of our attack for the `ees449ep1` parameters ($N = 449$, $df = 134$, $dg = 149$, $p = 3$, and $q = 2048$). Figure 1b shows the success

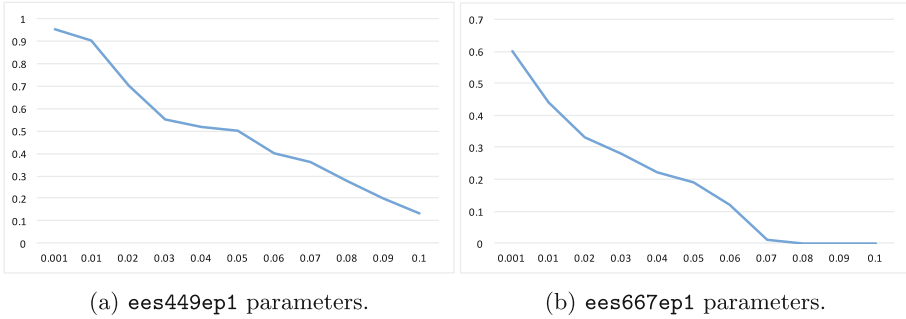


Fig. 1. Success rate of our algorithm (y -axis) against β (x -axis) for a fixed $\alpha = 0.001$, using Class II intervals.

rate for the `ees677ep1` parameters ($N = 677$, $\text{df} = 157$, $\text{dg} = 225$, $p = 3$, and $q = 2048$).

It can be seen from the two figures that the success rate is acceptably high for small values of β , but rapidly reduces as β is increased in size. Increasing the size of r (and therefore the search interval $[\max - r\sigma, \max]$) would improve the success rate at the cost of increased running time. For $r = 0.01$, we saw running times on the order of minutes to hours. There were a few simulations with very high running times; these were aborted after 1 day of computation. We observed this behaviour in particular for high values of β . In this case, the number of tasks, 3, the number of chunks, 400, and the nature itself of what was considered a suitable candidate (number of 1's and -1 's) made it hard to predict the number of candidates in a given interval. So searching over a given interval is done somewhat “blindly”, in the sense that searching over the interval $[\max - 0.01\sigma, \max]$ will not behave in a consistent manner in terms of the number of candidates found (and hence the running time needed).

5.3 Results for the `tbuktu` Java Implementation

Due to the additional structure of private keys compared to the `ntru-crypto` implementation, we focussed a greater experimental effort on the `tbuktu` Java implementation.

5.3.1 Counting Candidates and Estimating Running Times

Because of the nature of the log-likelihood function employed to calculate scores, each of the six lists \mathcal{C}_i output by Phase I of the attack will have many repeated score values. This enables us to efficiently compute the *number* of candidates that Phase II of the attack will consider in any given interval $[a, b]$. To do this, we run a modified version of Phase II in which the lists \mathcal{C}_i are replaced by “reduced” lists which eliminate chunk candidates having repeated score values, and include the counts (numbers) of such candidates along with their common score. By simultaneously computing the sums of scores and *products* of counts

on these reduced lists, we can compute the total number of candidates that will have a given score, over all possible scores in any chosen interval.

Because the size of each reduced list is less than 10 on average in our experiments (when `lsize` is up to 1024), we obtain a very efficient algorithm for counting the number of candidates in any given interval that our Phase II search algorithm would need to consider. We can combine this counting algorithm with the average time needed to generate and consider each candidate to get estimates for the total running time that our algorithm would encounter for a given choice of interval. We can then also compute the expected success probability and estimated running time (for the given number of candidates or given interval considered) without actually running the full Phase II search algorithm.

5.3.2 Parameters

The encryption parameters used for running the simulations are APR2011_439_FAST ($N = 439$, $p = 3$, $q = 2048$, $df1 = 9$, $df2 = 8$, $df3 = 5$, `sparse = true`, `fastP = true` so that $\mathbf{t} = \mathbf{f}_1\mathbf{f}_2 + \mathbf{f}_3$, and $\mathbf{f}_1 \in T(9, 9)$, $\mathbf{f}_2 \in T(8, 8)$, $\mathbf{f}_3 \in T(5, 5)$).

5.3.3 Results – Complete Enumeration

In our experiments, we set `lsize` to 2^r for Phase I, for $r = 8, 9, 10$. Thus six candidate lists each of size 2^r will be obtained from Phase I. Let p_i denote the probability that the correct candidate is actually found in the i -list; p_i will be a function of r . It follows that the probability that our Phase II algorithm outputs the correct private key when performing a *complete* enumeration over all 2^{6r} candidate keys is given by $p = \prod_{i=1}^6 p_i$. This simple calculation gives us a way to perform simulations to estimate the expected success rate of our overall algorithm (Phase I and Phase II) without actually executing the expensive Phase II. We simply run many simulations of Phase I for the given value of `lsize` (each simulation generating a fresh private key and perturbing it according to α, β), and, after each simulation, test whether the correct chunks of the private keys are to be found in the lists.

Figure 2 shows the success rates for complete enumeration for values of `lsize` = 2^r for $r \in \{8, 9, 10\}$. As expected, the greater the value of `lsize`, the higher the success rate for a fixed α and β . Also note that when the noise is high (for example $\alpha = 0.09$ and $\beta = 0.09$), the success rate drops to zero. This is expected since it is likely that at least one chunk of the private key will not be included in the corresponding list coming out of Phase I when the noise levels are high, at which point Phase II inevitably fails.

Note that each data point in this figure (and all figures in section) were obtained using 100 simulations. Note that the running times for Phase I are very low in average (≤ 50 ms), since that phase consists of calling the OKEA for each one of the six lists with `lsize` in the set $\{256, 512, 1024\}$.

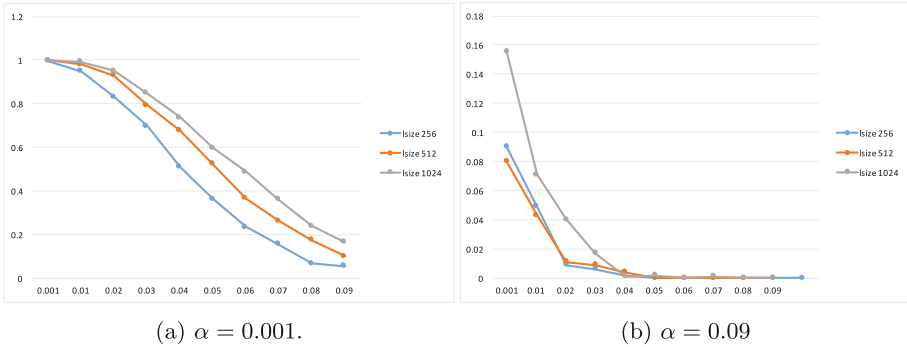


Fig. 2. Expected success rate for a full enumeration for $\alpha = 0.001, 0.09$. The y -axis represents the success rate, while the x -axis represents β .

5.3.4 Results – Partial Enumeration

Here, we exploit our counting algorithm to estimate success rates as a function of the total number of keys considered, K . Specifically, given a value K , and an interval type (I or II), we can set W accordingly so that the right number of keys will be considered. Since we can easily estimate the speed at which individual keys can be assessed, we can also use this approach to control the total running time of our algorithms.

Figure 3 shows how the success rate of our algorithm varies for different values of `lsize`, focussing on Class I intervals. We observe the same trends as for full enumeration, i.e. the greater is `lsize`, the higher is the success rate for a fixed α and β . Also, for larger values of (α, β) , the success rate drops rapidly to zero.

Figure 4 shows the success rates for a complete enumeration and partial enumerations with 2^{30} keys and 2^{40} keys, for both Class I and Class II intervals.

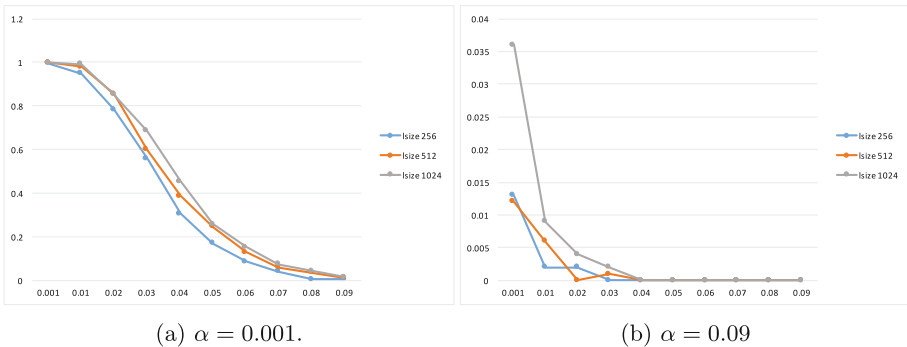


Fig. 3. Success rate for enumeration with 2^{40} keys over a Class I interval for $\alpha = 0.001, 0.09$, for different values of `lsize`. The y -axis represents the success rate, while the x -axis represents β .

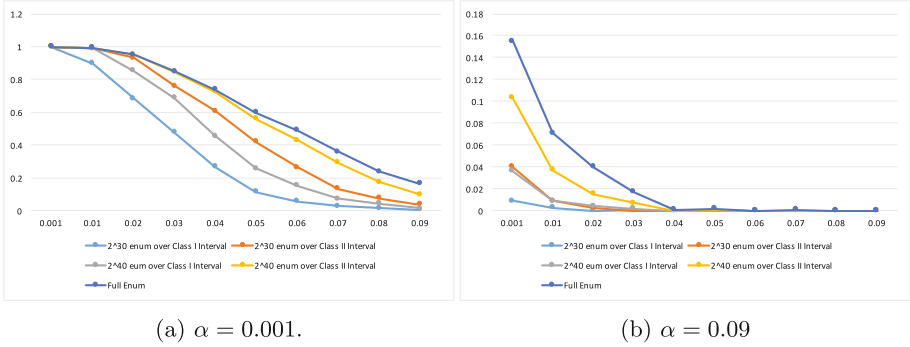


Fig. 4. Success rates for full enumeration, and partial enumeration with 2^{30} keys, 2^{40} keys for $\alpha = 0.001, 0.09$ and with `lsize` = 1024. The y -axis represents the success rate, while the x -axis represents β .

As expected, the success rate for a full enumeration is greater than for the partial enumerations (but note that a full enumeration here would require the testing of up to 2^{60} keys, which may be a prohibitive cost). Note that the closest success rate to the success rate of a full enumeration is achieved with partial enumerations with 2^{40} keys over a Class II interval, and that partial enumerations over Class I intervals perform poorly, in the sense that their success rates are even dominated by the success rate of enumerations with 2^{30} keys over Class II intervals. The superiority of Class II intervals is in-line with the intuition that testing high log-likelihood candidates for correctness is better than examining average log-likelihood ones.

5.3.5 Running Times

From our experiments, we find that our code is able to test up to 1200 candidates per millisecond per core during Phase 2. This value may vary in the range 700–1200 when there are multiples tasks running. The reason for this variation may be the cost associated with the Java virtual machine (particularly, its garbage collector). Using only a single core, an enumeration of 2^{30} (2^{40}) candidate keys will take about 14 min (10 days, respectively).

6 Conclusions

We have initiated the study of cold boot attacks for the NTRU public key encryption scheme, likely to be an important candidate in NIST’s forthcoming post-quantum standardisation process. We have proposed algorithms for this problem, with particular emphasis on two existing NTRU implementations and two private key formats. We have experimented with the algorithms to explore their performance for a range of parameters, showing how algorithms developed for enumerating keys in side-channel attacks can be successfully applied to the problem.

Our attacks do not exploit the underlying mathematical structure of the NTRU scheme. It would be interesting to explore whether our techniques can be combined with other approaches, such as lattice-reduction, to further improve performance. We also focussed mainly on the two available Java implementations. It would be interesting to extend our work to consider the `ntru-encrypt C` implementation which uses packing techniques to reduce the private key size. This seems challenging because of the corresponding increase in information density for these formats; however, there is still some redundancy in the second of the two formats because of the ordering of indices. It is an interesting open problem to find ways to exploit this redundancy. Implementations of NTRU may also compute additional private key values, for example the inverse of $\mathbf{f} \bmod p$, in order to speed up decryption operations. Thus these extra values might be available in a cold boot attack. Finding methods for exploiting this additional redundancy would be of interest.

Acknowledgements. The research of Villanueva-Polanco was supported by Colciencias.

References

1. Albrecht, M., Cid, C.: Cold boot key recovery by solving polynomial systems with noise. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 57–72. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21554-4_4
2. Albrecht, M.R., Orsini, E., Paterson, K.G., Peer, G., Smart, N.P.: Tightly secure ring-LWE based key encapsulation with short ciphertexts. *Cryptology ePrint Archive, Report 2017/354* (2017). <http://eprint.iacr.org/2017/354>
3. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU prime. *Cryptology ePrint Archive, Report 2016/461* (2016). <http://eprint.iacr.org/2016/461>
4. Bogdanov, A., Kizhvatov, I., Manzoor, K., Tischhauser, E., Witteman, M.: Fast and memory-efficient key recovery in side-channel attacks. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 310–327. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31301-6_19
5. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: CRYSTALS - Kyber: a CCA-secure module-lattice-based KEM. *Cryptology ePrint Archive, Report 2017/634* (2017). <http://eprint.iacr.org/2017/634>
6. David, L., Wool, A.: A bounded-space near-optimal key enumeration algorithm for multi-subkey side-channel attacks. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 311–327. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_18
7. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold boot attacks on encryption keys. In: van Oorschot, P.C. (ed.) Proceedings of the 17th USENIX Security Symposium, San Jose, CA, USA, 28 July–1 August 2008, pp. 45–60. USENIX Association (2008)
8. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM* **52**(5), 91–98 (2009)

9. Henecka, W., May, A., Meurer, A.: Correcting errors in RSA private keys. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 351–369. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_19
10. Heninger, N., Shacham, H.: Reconstructing RSA private keys from random key bits. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 1–17. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_1
11. Hoffstein, J., Howgrave-Graham, N., Pipher, J., Whyte, W.: Practical lattice-based cryptography: NTRUEncrypt and NTRUSign. In: Nguyen, P.Q., Vallée, B. (eds.) The LLL Algorithm - Survey and Applications. Information Security and Cryptography, pp. 349–390. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-02295-1_11
12. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ring-based public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0054868>
13. Huang, Z., Lin, D.: A new method for solving polynomial systems with noise over \mathbb{F}_2 and its applications in cold boot key recovery. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 16–33. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_2
14. Kamal, A.A., Youssef, A.M.: Applications of SAT solvers to AES key recovery from decayed key schedule images. In: Savola, R., Takesue, M., Falk, R., Popescu, M. (eds.) Fourth International Conference on Emerging Security Information Systems and Technologies, SECURWARE 2010, Venice, Italy, 18–25 July 2010, pp. 216–220. IEEE Computer Society (2010)
15. Lee, H.T., Kim, H.T., Baek, Y.-J., Cheon, J.H.: Correcting errors in private keys obtained from cold boot attacks. In: Kim, H. (ed.) ICISC 2011. LNCS, vol. 7259, pp. 74–87. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31912-9_6
16. Martin, D.P., Mather, L., Oswald, E., Stam, M.: Characterisation and estimation of the key rank distribution in the context of side channel evaluations. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 548–572. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_20
17. Martin, D.P., O’Connell, J.F., Oswald, E., Stam, M.: Counting keys in parallel after a side channel attack. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 313–337. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_13
18. Paterson, K.G., Polychroniadou, A., Sibborn, D.L.: A coding-theoretic approach to recovering noisy RSA keys. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 386–403. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_24
19. Poettering, B., Sibborn, D.L.: Cold boot attacks in the discrete logarithm setting. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 449–465. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16715-2_24
20. Veyrat-Charvillon, N., Gérard, B., Renauld, M., Standaert, F.-X.: An optimal key enumeration algorithm and its application to side-channel attacks. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 390–406. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_25

Differential Cryptanalysis of 18-Round PRIDE

Virginie Lallemand and Shahram Rasoolzadeh^(✉)

Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany
{virginie.lallemand,shahram.rasoolzadeh}@rub.de

Abstract. The rapid growth of the Internet of Things together with the increasing popularity of connected objects have created a need for secure, efficient and lightweight ciphers. Among the multitude of candidates, the block cipher PRIDE is, to this day, one of the most efficient solutions for 8-bit micro-controllers. In this paper, we provide new insights and a better understanding of differential attacks of PRIDE. First, we show that two previous attacks are incorrect, and describe (new and old) properties of the cipher that make such attacks intricate. Based on this understanding, we show how to properly mount a differential attack. Our proposal is the first single key differential attack that reaches 18 rounds out of 20. It requires 2^{61} chosen plaintexts and recovers the 128-bit key with a final time complexity of $2^{63.3}$ encryptions, while requiring a memory of about 2^{35} blocks of 64 bits.

Keywords: Block cipher · PRIDE · Differential cryptanalysis

1 Introduction

We are currently facing a growing need for secure and efficient cryptographic primitives that aim to protect the myriad of resource-constrained devices that are more and more part of our daily lives.

Most popular examples of such targeted devices of the Internet of Things include RFID tags and nodes in sensor networks. For the latter, one of the preferred platforms are 8-bit micro-controllers. Ciphers dedicated to this platform require to be lightweight and software-oriented, that is, in addition to being secure will only require a small program memory and have a small execution time. Examples of ciphers proposed to meet these needs include SEA [11], KLEIN [5], ITUbee [7], PRIDE [1] and the Feistel ciphers designed by the National Security Agency (SIMON and SPECK [2]). Among the academic proposals, the substitution permutation network (SPN) PRIDE proposed by Albrecht et al. at Crypto 2014 is the most efficient, result that sources from the designers' careful analysis of linear layers that reach interesting trade-off between security and efficiency.

Previous works on PRIDE include a side-channel attack presented at CRiSIS 2016 [9]. In the black box scenario, Dinur presented at Eurocrypt 2015 [4] a new cryptanalytic time-memory-data trade-off, while Guo et al. [6] gave observations on the impact of increasing the number of rounds of the cipher. The more

powerful attacks published to date are a related-key differential attack of the full cipher [3] by Dai and Chen, and two differential attacks on 18 [16] and 19 [15] rounds out of 20. Quoting from the specification document¹, the related key attack is out of scope: “PRIDE *does not claim any resistance against related-key attacks (and actually can be distinguished trivially in this setting)*”, so the best type of attack appears to be single key differential attack.

In this paper we provide insight on the resistance of PRIDE against this type of attack and give a twofold contribution: first, we show that the two previous attacks ([15, 16]) are erroneous — even when taking into account the corrections proposed by [13] — due to a miscalculation of the known bits and second we show how to correctly mount a differential cryptanalysis to attack 18 rounds of PRIDE.

Our attack requires 2^{61} chosen plaintexts and the equivalent of $2^{63.3}$ encryptions. Since the security claim of the designers is that the product of data and time complexity cannot be smaller than 2^{127} , our proposal is a valid attack of the cipher reduced to 18 rounds.

The paper is organized as follows. Next section gives a short description of the block cipher PRIDE and introduces our notations. Then, we start our study with a section reporting old and new properties of PRIDE Sbox and key schedule. In Sect. 4, we describe our first contribution by disclosing why the two previous differential cryptanalyses of PRIDE fail to recover the key, even when the flaws spotted in previous works are corrected. We then put into practice our comprehension of PRIDE to build high probability differential characteristics (Sect. 5) and mount an 18-round differential attack in Sect. 6. The paper ends with a conclusion.

2 PRIDE Block Cipher

2.1 Description of PRIDE

PRIDE [1] is a lightweight block cipher proposed at Crypto 2014 by Albrecht, Driessen, Kavun, Leander, Paar and Yalçın. The cipher follows an SPN structure and benefits from an extensive analysis of secure and efficient linear layers, presented in the same article. It is software-oriented and reaches notable performance figures when implemented on 8-bit micro-controllers.

Round Function. PRIDE uses 64-bit blocks and 128-bit keys and makes use of the FX construction [8] in the following way: the first 64 bits of the master key k , denoted k_0 , is used as pre- and post-whitening key, while the other half k_1 is used to compute the round keys. In the following, we denote the whitening key by K_0 and the round key of round i by K_i , $1 \leq i \leq 20$ (see Fig. 1).

PRIDE encryption routine is made of 20 rounds. The first 19 rounds are identical and denoted by \mathcal{R} , while the last one does not contain the linear layer and is denoted by \mathcal{R}' . The cipher ends (resp. starts) with the application of a

¹ Sect. 5.5 of [1].

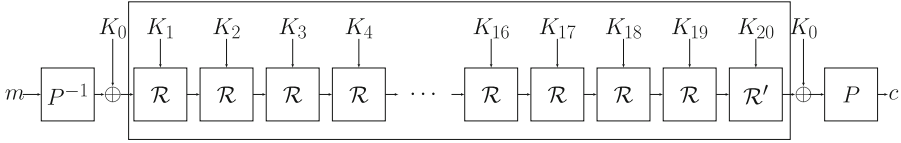


Fig. 1. Overall structure of PRIDE block cipher.

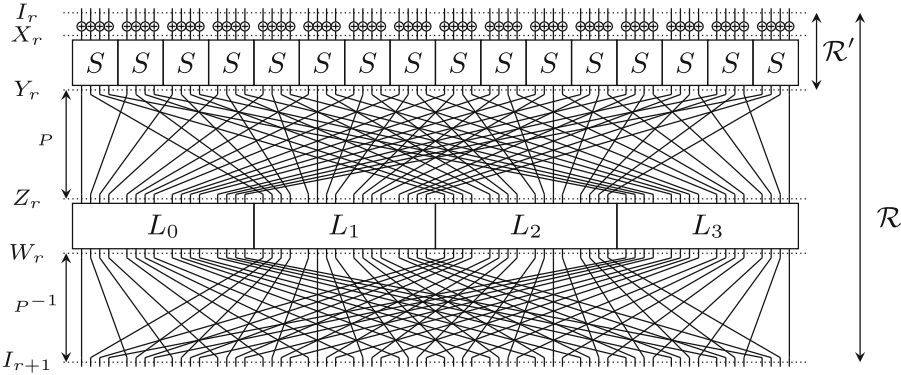


Fig. 2. \mathcal{R} and \mathcal{R}' round functions of PRIDE. The naming conventions used for the intermediate states are detailed in Table 2.

bit-permutation (resp. its inverse) for bit-sliced implementation reasons. Since these operations can easily be inverted, what we call in the following *plaintext* and *ciphertext* are the states before (resp. after) the first (resp. last) whitening operation. The cipher is based on the following operations², combined as depicted in Fig. 2:

- A key addition layer,
- An Sbox layer, which consists in applying the same 4×4 Sbox S (given in Table 1) to each nibble (group of 4 bits) of the state,
- A linear layer, combining:
 - The application of bit permutations P and P^{-1} ,
 - The application of matrices, more precisely the application of matrix L_i , $i = 0, \dots, 3$ to the i^{th} 16-bit word of the state.

Table 1. Definition of the Sbox of PRIDE.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	0	4	8	f	1	5	e	9	2	7	a	c	b	d	6	3

² Due to space limitations, we refer to [1] for the details of the linear layer.

Key-Schedule. The round keys of PRIDE are 64-bit words given by $K_i = P^{-1}(f_i(k_1))$ ($1 \leq i \leq 20$) where $f_i(k_1)$ is:

$$f_i(k_1) = k_{1_0} || g_i^{(0)}(k_{1_1}) || k_{1_2} || g_i^{(1)}(k_{1_3}) || k_{1_4} || g_i^{(2)}(k_{1_5}) || k_{1_6} || g_i^{(3)}(k_{1_7})$$

k_{1_i} , $0 \leq i < 8$, is byte number i of k_1 and the g_i functions are given by:

$$\begin{aligned} g_i^{(0)}(x) &= (x + 193i) \bmod 256, & g_i^{(1)}(x) &= (x + 165i) \bmod 256, \\ g_i^{(2)}(x) &= (x + 81i) \bmod 256, & g_i^{(3)}(x) &= (x + 197i) \bmod 256. \end{aligned}$$

2.2 Notations

To ease comprehension of the remainder of the paper, we use the same notation as in the two previous differential attacks on PRIDE ([15, 16]). These notations are recalled in Table 2. In order to remain consistent with it, we also start counting bits from 1, and more particularly we denote by (x_1, x_2, x_3, x_4) the binary decomposition of the nibble x , where x_1 is its most significant bit.

Table 2. Notations.

Symbol	Definition
I_r	input state of r -th round
X_r	state after key addition of r -th round
Y_r	state after the Sbox layer of r -th round
Z_r	state after the application of P of r -th round
W_r	state after the matrices layer of r -th round
ΔS	<i>xor</i> difference of the state S
$S_r[i]$	i -th nibble of the state S_r
$S_r^j[i]$	j -th bit of the i -th nibble of S_r

3 Properties of PRIDE Components

In this section, we present important properties of the Sbox and of the Key-Schedule that impact a differential attack of PRIDE. These properties are crucial to understand the mistakes made in the previous differential cryptanalyses as well as to get the techniques used in our new attack.

3.1 Sbox Properties

We start by recalling the component functions of the Sbox:

Definition 1 (Component functions of PRIDE Sbox). *If we denote $x = (x_1, x_2, x_3, x_4)$ the input nibble of the Sbox, then the expressions of the corresponding output nibble $S(x) = y = (y_1, y_2, y_3, y_4)$ is given by:*

$$\begin{cases} y_1 = x_1x_2 \oplus x_3 \\ y_2 = x_2x_3 \oplus x_4 \\ y_3 = x_1x_2x_3 \oplus x_1x_2x_4 \oplus x_2x_3 \oplus x_3x_4 \oplus x_1 \\ y_4 = x_1x_2x_4 \oplus x_1x_4 \oplus x_2x_3 \oplus x_3x_4 \oplus x_2 \end{cases}$$

We can remark that y_1 and y_2 depend only on 3 bits out of 4 of the input and that only two of the input bits are involved in the degree 2 monomials. This remark turns useful in our attack since it implies that instead of requiring a complete nibble to get the value of bit number 1 or 2 we only need the value of 3 bits. Note that since PRIDE Sbox is an involution these properties also hold for its inverse.

What’s more, this observation impacts the possible differential transitions of the Sbox, a property that was formalized by Tezcan in [12] and applied to PRIDE in [13].

Definition 2 (undisturbed bit [12]). *For a specific input difference of an S-box, if some bits of the output difference remain invariant, then we call such bits undisturbed.*

For instance, if the input difference of PRIDE Sbox is equal to 8 (1000), its output difference is of the form ?0?? (see [13]).

In [14], Tezcan and Özbudak introduced the notion of differential factor, that plays a role in the number of key bits one can recover and on the time complexity:

Definition 3 (differential factor [14]). *Let S be a function from \mathbb{F}_2^n to \mathbb{F}_2^m . For all $x, y \in \mathbb{F}_2^n$ that satisfy $S(x) \oplus S(y) = \mu$, if we also have $S(x \oplus \lambda) \oplus S(y \oplus \lambda) = \mu$, then we say that the S-box has a differential factor λ for the output difference μ . (i.e. μ remains invariant for λ).*

3.2 Key-Schedule Properties

We introduce here a property of the key schedule that allows to reduce the number of key-guesses required in our attack:

Property 1 (Difference between round keys). The binary difference between two round keys K_i and K_j for i and j of different parity is given by the following expression, where ‘?’ represents an unknown bit:

$$K_i \oplus K_j = (00000000|00000000|00000000|00000000|????????|????????|????????|????1111)$$

Also, the difference between K_i and K_ℓ for i and ℓ of same parity³ is given by:

$$K_i \oplus K_\ell = (00000000||00000000||00000000||00000000||????????||????????||????????||????0000).$$

Proof. The first relation results from the definition of the round key, from which we obtain that:

$$K_i \oplus K_j = P^{-1}(00000000||????????1||00000000||????????1||00000000||????????1||00000000||????????1)$$

where the differences of ‘1’ in bit 16, 32, 48 and 64 of $P(K_i \oplus K_{i+1})$ are easily explained by the fact that i and j have different parities and that the values added to k_1 in g functions are odd. The second relation results from the fact that i and ℓ have the same parity.

As described later, we select our characteristic so that when checking the active Sboxes we have common bits so less guesses to make.

4 Previous Differential Attacks on PRIDE

Two single key differential attacks ([15, 16]) have been published prior to our work. In [13], Tezcan et al. show that the complexities of these attacks are miscomputed due to the oversight of the impact of differential factor and propose a correction. Their patch mainly results in an increase of the final time complexity.

In this section, we show that there are more problems in [15, 16] than the ones reported in [13] and that consequently the proposed patches are insufficient. The problem we disclose and that is common to both attacks is that the attacker misses information to compute the required internal state bits.

4.1 18-Round Differential Attack of Zhao et al.

In [16], Zhao et al. proposed an attack on 18-round PRIDE. They use a 15-round characteristic⁴ of probability 2^{-58} and add one round to the top and two rounds to the bottom. Their attack procedure starts by eliminating some wrong pairs by looking at the ciphertext difference. Then, they guess 10 nibbles of the whitening key K_0 – namely $K_0[1, 2, 3, 5, 6, 7, 10, 11, 14, 15]$ – in order to be able to check that the differences at the input of the corresponding Sboxes of round 18 have the right form (see Table 3).

They next introduce K'_{18} , a key that is equivalent to the last round key K_{18} and is given by: $(M \circ P)^{-1}(f_{18}(k_1))$. They make a guess on $K'_{18}[6, 10, 14]$ in order to be able to compute the difference entering Sbox number 6, 10 and 14 of penultimate round and access the corresponding sieve.

This attack suffers from several problems: first, as noted in [15] and later in [13], the authors omitted to take into account the undisturbed bits. In addition to that,

³ Note that simple relations can also be found between other keys; K_i and K_{i+16} for instance.

⁴ It corresponds to what we name in next section the first characteristic of type (I, a) , see Table 5.

[13] reveals that the 6 Sbox differences that are involved in the attack are differential factors (namely $\lambda = \mu = 8$), which implies that the attacker cannot obtain information on 6 key bits. Quoting [13], this error results in the fact that “the correct time complexity of this attack is 2^{70} 18-round Pride encryptions, not 2^{66} ”.

The new problem we spotted is a miscalculation of the known bits of the internal states. Namely: to compute the difference entering Sbox number 6, 10 and 14 of penultimate round we need the value of $Y_{17}[6, 10, 14]$, but it is impossible to determine the two middle bits of any of these 3 nibbles. This phenomenon appears clearly if we look at Table 3, where we have depicted the bits of rounds 17 that can be computed from the ciphertext given the key guesses on K_0 . We clearly see that the 3 highlighted nibbles $Y_{17}[6]$, $Y_{17}[10]$ and $Y_{17}[14]$ are not completely determined.

Table 3. Analysis of 18-round differential attack of PRIDE by Zhao et al. [16]. All the bit values that are computable are depicted with a ‘1’, while other bits are shown by ‘0’.

$P = I_1$	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111
$K_0 \oplus K_1$	0000 0000 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000
X_1	0000 0000 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000
Y_1	0000 0000 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000
...				...			
X_{17}	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000
K'_{18}	0000 0000 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000	0000 1111 0000 0000
Y_{17}	0000 1001 1001 0110	0000 1001 1001 0110	0000 1001 1001 0110	0000 1001 1111 0000	0000 1001 1111 0000	0000 1001 1111 0000	0000 1001 1111 0000
Z_{17}	0110 0110 0110 0110	0001 0001 0010 0010	0001 0001 0010 0010	0001 0001 0010 0010	0001 0001 0010 0010	0110 0110 0110 0110	0110 0110 0110 0110
W_{17}	1110 1110 0110 0110	1110 1110 0110 0110	1110 1110 0110 0110	1110 1110 0110 0110	1110 1110 0110 0110	1110 1110 0110 0110	1110 1110 0110 0110
X_{18}	1111 1111 1111 0000	1111 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000
Y_{18}	1111 1111 1111 0000	1111 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000
K_0	1111 1111 1111 0000	1111 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000	0000 1111 1111 0000
C	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111

As it is, the sieve offered by these 3 Sboxes cannot be accessed, so each possible value for the 52 bits of key would be suggested 2^7 times in average, and the right value would not be distinguishable. Consequently, the attack fails.

The attack of Zhao et al. has high requirements (2^{60} messages and 2^{66} encryptions), so taking into account the correction from the differential factors already leads to an attack that does not break the security claim ($2^{60} \times 2^{70} > 2^{127}$). In addition to that, correcting the problem we spotted in a straightforward manner would require to make more guesses on K_0 , that is to guess 22 bits of the nibbles $K_0[4, 8, 9, 12, 13, 16]$, so clearly fixing Zhao et al. paper does not lead to an attack that threaten the cipher.

The authors’ confusion comes probably from the fact that the linear layer is not an involution. In the case it was, knowing the bits they name would have been enough to access the active Sboxes in round 17. Unfortunately, L_1 and L_2 do not define involutions so more bits are required to find the output of the 3 active Sboxes of round 17.

4.2 19-Round Differential Attack by Yang et al.

There is a similar mistake in the 19-round attack described by Yang et al. in [15].

They use a 15-round characteristic⁵ and expand it two rounds to the plaintext side and two rounds to the ciphertext side. After discarding pairs that for sure do not follow the characteristic, they guess nine nibbles of key in plaintext side, namely $(K_0 \oplus K_1)[1, 2, 3, 5, 7, 9, 10, 13, 14]$, and partially encrypt the plaintext pairs through the first Sbox layer. On the ciphertext side, they guess the seven nibbles 1, 2, 5, 8, 9, 10 and 13 of K_0 and partially decrypt the ciphertext pairs through the last Sbox layer. They next claim that they can also recover nibbles number 5 and 9 of K_2 and nibbles 5 and 9 of $K'_{19} = (M \circ P)^{-1}(f_{19}(k_1))$.

Similarly to the 18-round attack discussed in previous section and as described in [13], this attack uses difference transitions that are differential factors, meaning that it fails to recover 4 bits of the key (each most significant bit of nibbles number 5 and 9 of K_2 and nibbles 5 and 9 of K'_{19}).

Moreover, as in previous section, we note that there is a problem upstream to that one. Namely, the authors simply cannot compute the desired Sbox transitions given their guesses: they lack information to compute the two middle bits of $I_2[5, 9]$ and $Y_{18}[5, 9]$. Consequently, they cannot obtain information on these four nibbles, and the right key cannot be identify (even if the correction given by Tezcan et al. is applied).

The detail of which bits are computable in first and last two rounds is provided in Table 4.

Table 4. Analysis of 19-round differential attack of PRIDE by Yang et al. [15]. We use the same notation as before and depict known bits with ‘1’.

$P = I_1$	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111
$K_0 \oplus K_1$	1111 1111 1111 0000	1111 0000 1111 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000
X_1	1111 1111 1111 0000	1111 0000 1111 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000
Y_1	1111 1111 1111 0000	1111 0000 1111 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000
Z_1	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100	1110 1010 1100 1100
W_1	1000 1100 1000 1000	0100 0000 0100 0000	0000 0100 0000 0100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100	1000 1000 1000 1100
I_2	1001 0100 0000 0000	1001 1010 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000	1001 0100 0000 0000
K_2	0000 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000
X_2	0000 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000	1001 0000 0000 0000
Y_2	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000
...														
X_{18}	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000
K'_{19}	0000 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000	1111 0000 0000 0000
Y_{18}	1001 0000 0100 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000	1001 0000 0010 0000
Z_{18}	1000 1000 1000 1000	0010 0000 0000 0100	0000 0010 0100 0000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000	1000 1000 1000 1000
W_{18}	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000	1100 1001 1100 1000
X_{19}	1111 1111 0000 0000	1111 0000 0000 1111	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000
Y_{19}	1111 1111 0000 0000	1111 0000 0000 1111	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000
K_0	1111 1111 0000 0000	1111 0000 0000 1111	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000	1111 1111 0000 0000
C	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111

⁵ The fourth characteristic of type (II, a) given in Table 5.

The initial attack requires 2^{62} chosen plaintexts and 2^{63} 19-round encryptions. To correct the errors we spotted, it is necessary to significantly increase the number of key guesses. We need the value of 24 bits of nibbles number 3, 4, 7, 11, 12, 15, 16 of K_0 to be able to treat the 2 Sboxes of penultimate round, while we need the value of $(K_0 \oplus K_1)[12, 16]$ and 3 bits of $(K_0 \oplus K_1)[6]$ to access the 2 Sboxes of round 2. Clearly, the straightforward correction does not lead to a correct attack since the time complexity explodes.

5 Differential Characteristics for PRIDE

5.1 1 and 2-Round Iterative Differential Characteristics

As already shown in [15,16], there are 56 high-probability iterative characteristics on 1 and 2 rounds of PRIDE, each activating only 4 Sboxes on 2 rounds whose both input and output differences are equal to 8. Hence, the probability of any of these iterative characteristics is equal to $(2^{-2})^4 = 2^{-8}$. The 56 possible input/output differences are given in Table 5, where they are grouped according to the number of active Sboxes in the first round (line *I*, *II* or *III*) and to the index of the first active Sbox in the input difference (column *a*, *b*, *c* and *d*). Note that all type *II* characteristics are iterative on 1 round while the others are iterative on 2 rounds.

Table 5. Hexadecimal value of all the 1 and 2-round iterative differential characteristics of PRIDE. The characteristic used in our attack is highlighted.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>I</i>	8000 0000 0000 0000	0800 0000 0000 0000	0080 0000 0000 0000	0008 0000 0000 0000
	0000 8000 0000 0000	0000 0800 0000 0000	0000 0080 0000 0000	0000 0008 0000 0000
	0000 0000 8000 0000	0000 0000 0800 0000	0000 0000 0080 0000	0000 0000 0008 0000
	0000 0000 0000 8000	0000 0000 0000 0800	0000 0000 0000 0080	0000 0000 0000 0008
<i>II</i>	8000 8000 0000 0000	0800 0800 0000 0000	0080 0080 0000 0000	0008 0008 0000 0000
	8000 0000 8000 0000	0800 0000 0800 0000	0080 0000 0080 0000	0008 0000 0008 0000
	8000 0000 0000 8000	0800 0000 0000 0800	0080 0000 0000 0080	0008 0000 0000 0008
	0000 8000 8000 0000	0000 0800 0800 0000	0000 0080 0080 0000	0000 0008 0008 0000
	0000 8000 0000 8000	0000 0800 0000 0800	0000 0080 0000 0080	0000 0008 0000 0008
0000 0000 8000 8000	0000 0000 0800 0800	0000 0000 0080 0080	0000 0000 0008 0008	
<i>III</i>	0000 8000 8000 8000	0000 0800 0800 0800	0000 0080 0080 0080	0000 0008 0008 0008
	8000 0000 8000 8000	0800 0000 0800 0800	0080 0000 0080 0080	0008 0000 0008 0008
	8000 8000 0000 8000	0800 0800 0000 0800	0080 0080 0000 0080	0008 0008 0000 0008
	8000 8000 8000 0000	0800 0800 0800 0000	0080 0080 0080 0000	0008 0008 0008 0000

5.2 14-Round Differential Characteristics

Repeating any of the iterative characteristics of Table 5 gives a 14-round characteristic of probability 2^{-56} . To find out if there are other 14-round characteristics with similar or better probability, we searched for characteristics with up to 3 active Sboxes in each round. Our program returned 168 (new) 14-round characteristics of probability 2^{-56} . Unfortunately, these characteristics are less advantageous than the iterative ones since when we propagate them with probability 1 in the forward and backward direction they activate more Sboxes.

Assume then that we use a 14-round characteristic (built from one of Table 5) between round 3 and 17 of the cipher. By inverting the linear layer, we compute ΔY_2 from ΔI_3 , thus capturing 56 pairs $(\Delta Y_2, \Delta X_{17})$ that hold with probability 2^{-56} . In addition to that, a 14-round characteristic defines a limited number of possible differences for ΔI_2 and ΔY_{18} that can be computed from the distribution table of the Sbox (see the full version of the paper ([10]) or Table 2 of [16]). Namely, each active Sbox of ΔI_2 and ΔY_{17} can only take 4 values, so we obtain that ΔI_2 and ΔY_{17} can respectively take 4^{n_2} and $4^{n_{17}}$ values, where n_i represents the number of active Sboxes in round i .

6 Differential Cryptanalysis of 18-Round PRIDE

This section describes our 18-round differential cryptanalysis of PRIDE. We start by exposing a differential property of PRIDE Sbox and then show how to use it in an attack to easily find information on key bits. We then detail the complexities of our attack.

6.1 PRIDE Sbox Properties for Our Differential Characteristics

As discussed in Sect. 5.2, the difference transitions made by the Sboxes of round 2 and 17 are either from 8 to 2, 3, 8 or a or from 2, 3, 8 or a to 8. For these configurations, the following property holds:

Property 2 (Relations defined by difference transitions of the Sbox).

If two Sbox inputs differ by 2 (respectively 3, 8 or a) and lead to an output difference of 8 then the following relations hold:

$$\begin{aligned} S(x) \oplus S(x \oplus 2) &= S(x_1 x_2 x_3 x_4) \oplus S(x_1 x_2 \bar{x}_3 x_4) = 8 \Rightarrow x_2 = 0, x_4 = 0 \\ S(x) \oplus S(x \oplus 3) &= S(x_1 x_2 x_3 x_4) \oplus S(x_1 x_2 \bar{x}_3 \bar{x}_4) = 8 \Rightarrow x_2 = 1, x_3 = x_4 \\ S(x) \oplus S(x \oplus 8) &= S(x_1 x_2 x_3 x_4) \oplus S(\bar{x}_1 x_2 x_3 x_4) = 8 \Rightarrow x_2 = 1, x_3 = \bar{x}_4 \\ S(x) \oplus S(x \oplus a) &= S(x_1 x_2 x_3 x_4) \oplus S(\bar{x}_1 x_2 \bar{x}_3 x_4) = 8 \Rightarrow x_2 = 0, x_4 = 1 \end{aligned}$$

Proof. The property results from the component functions (Definition 1).

In other words, if we are able to check that the input difference of an active Sbox is 2, 3, 8 or a and if we expect its output difference to be equal to 8 then we are able to deduce information on the value of the state entering this Sbox. Namely, we obtain the value of x_2 together with either the value of x_4 or a relation between x_4 and x_3 .

This observation implies that to check if an Sbox executes the right transition (so to have access to the corresponding filter of probability 2^{-2}) we only require information on (at most) 3 bits (bits 2, 3 and 4). This can be seen as a reinterpretation of the undisturbed bits of [12].

6.2 Overview of the Attack Procedure

In our attack, we use the 14-round differential path obtained by repeating the characteristic (iterative on 1 round) highlighted in Table 5 and extend it 2 rounds to the plaintext and 2 rounds to the ciphertext. This extension defines 4 rounds of key recovery that will allow us to recover 10 bits of key on the plaintext side and 10 bits of key on the ciphertext side.

The reasons why we decided to use this particular characteristic are the following. First, among the 224 (168 new and 56 previously found) characteristics we prefer the ones which imply less active Sboxes on the plaintext and ciphertext side, and consequently require to make less guesses and computations when checking that first and last round transitions are correct. This downsizes the set of candidate characteristics to 24 (8 of each type), each activating a total of 14 Sboxes on the plaintext and ciphertext side. Type *I* characteristics activate 9 Sboxes on plaintext side and 5 Sboxes on ciphertext side while for type *II* we have 7 active Sboxes on each side, and for type *III* the distribution is of 6 active Sboxes on plaintext side and 8 on ciphertext side.

Then, among the 24 possible characteristics, we prefer the ones that lead to smaller amount of possible differences⁶ when extending the characteristic with probability 1 in plaintext and ciphertext. We also take into account the number of key bits that we need to guess.

When looking for minimizing these parameters, both some of the characteristics of type *I* and type *II* seem good. Type *I* characteristics require 1 more key bit guess, but lead to less possible differences in plaintext and ciphertext. Eventually we prefer characteristics of type *II* since the memory size required to store a full structure is more reasonable.

This selection is further explained in the full version of the paper [10].

For the selected characteristic, and as explained in Sect. 5.2, ΔX_2 can take $4^2 = 16$ possible values, implying that there are also 16 possible values for ΔY_1 . As shown in Table 6, we have 7 active nibbles for the 16 possible differences of ΔY_1 (nibbles 4, 5, 6, 8, 9, 12 and 16) while the other 9 nibbles are always inactive. We use this property to reduce the necessary amount of data by building data structures: the messages we ask for encryption are organized as sets of $2^{4 \cdot 7} = 2^{28}$ plaintexts that are all equal in the corresponding 9 inactive nibbles and take all possible values in the above mentioned 7 nibbles. So in each structure there is about $2^{8 \cdot 7 - 1} = 2^{55}$ plaintext pairs that differ in the 7 nibbles of interest.

A pair from this structure has the correct difference in ΔY_2 if it reaches one of the 16 targeted values for ΔY_1 and then makes happen the correct transitions of Sbox 8 and 16 of the Sbox layer of round 2. The probability of this event can be computed as follows.

Probability that a pair takes one of the 16 targeted values in ΔY_1 . Consider a complete structure, that is the set made by all the messages with the same – fixed – value on our 9 inactive nibbles and taking all possible values on the other 7 nibbles. There are exactly 2^{28} such plaintexts. Since the key addition step

⁶ Here we look at real values instead of truncated differences.

defines a permutation, we still have 2^{28} messages differing on 7 nibbles after the addition of the whitening key K_0 and the first round key K_1 . The same reasoning applies to the Sbox layer: the messages in our structure take all possible values at the input of 7 Sboxes, so since PRIDE Sbox is a permutation the images of these messages still correspond to 2^{28} messages differing on the same positions. Consequently, when forming pairs of these messages, every possible non null difference on the 7 nibbles appears 2^{27} times, so in one structure exactly 16×2^{27} pairs out of 2^{55} are useful for our attack (i.e. a ratio of 1 out of 2^{24}).

Probability that one of the 16 differences in ΔY_1 leads to the correct ΔY_2 . If ΔY_1 is as required, the probability that the second round leads to the desired characteristic is equal to $(2^{-2})^2 = 2^{-4}$, which corresponds to the probability that the two active Sboxes of round 2 output a difference of 8 given an entering difference of 2, 3, 8 or a.

In sum, the total probability that one of our pairs follows the characteristic is equal to:

$$2^{-24} \cdot 2^{-4} \cdot 2^{-56} = 2^{-84}$$

which corresponds to realizing the correct transitions in round 1 and 2, following the 14-round characteristic⁷ and finally propagating with probability 1 in the last 2 rounds.

This indicates that we need to encrypt about $a \cdot 2^{84}$ plaintext pairs in order to obtain a pairs that follow the characteristic (also called *right pairs*). This amount can be obtained with $a \cdot 2^{84} \cdot 2^{-55} = a \cdot 2^{29}$ data structures i.e. with $a \cdot 2^{57}$ chosen plaintexts.

In the forward extension there are $4^2 = 16$ possible values for ΔY_{17} , so there are 16 possible values for ΔX_{18} . As shown in Table 6, these 16 possible values for ΔX_{16} define at most 7 active nibbles (nibbles 1, 3, 4, 8, 9, 12 and 16) while other 9 nibbles are always inactive. A common technique to filter out wrong pairs

Table 6. Differential extension of the 14-round characteristic used in our attack.

$\Delta P = \Delta X_1$	0000	0000	0000	????	????	????	0000	????	????	0000	0000	????	0000	0000	0000	????
ΔY_1	0000	0000	0000	?00?	00?0	00?0	0000	?00?	00?0	0000	0000	?00?	0000	0000	0000	?0??
ΔZ_1	000?	000?	000?	000?	0000	0000	0000	0000	0000	0000	??00	?000	000?	000?	000?	000?
ΔW_1	0000	000?	0000	000?	0000	0000	0000	0000	0000	0000	000?	0000	000?	0000	000?	000?
$\Delta I_2 = \Delta X_2$	0000	0000	0000	0000	0000	0000	0000	?0??	0000	0000	0000	0000	0000	0000	0000	?0??
ΔY_2	0000	0000	0000	0000	0000	0000	0000	1000	0000	0000	0000	0000	0000	0000	0000	1000
...																
$\Delta I_{17} = \Delta X_{17}$	0000	0000	0000	0000	0000	0000	0000	1000	0000	0000	0000	0000	0000	0000	0000	1000
ΔY_{17}	0000	0000	0000	0000	0000	0000	0000	?0??	0000	0000	0000	0000	0000	0000	0000	?0??
ΔZ_{17}	0000	000?	0000	000?	0000	0000	0000	0000	0000	000?	0000	000?	0000	000?	0000	000?
ΔW_{17}	000?	000?	000?	000?	0000	0000	0000	0000	?0??	0000	?00?	000?	000?	000?	000?	000?
$\Delta I_{18} = \Delta X_{18}$	00?0	0000	00?0	?0??	0000	0000	0000	?00?	00?0	0000	0000	?0??	0000	0000	0000	?0??
$\Delta Y_{18} = \Delta C$????	0000	????	????	0000	0000	0000	????	????	0000	0000	????	0000	0000	0000	????

⁷ Our experiments for up to 7 rounds showed that the probability of the differential matches the one of the characteristic.

consists in discarding pairs that have active Sboxes at any of these 9 positions. Given our harsh restrictions in terms of time complexity, we prefer considering a stronger filter which consists in checking that the difference observed in both plaintext and ciphertext differences are consistent with the 16 possible differences that can take Y_1 and X_{18} .

Once we generated enough messages and filtered them according to plaintext and ciphertext differences, we start making key guesses: we test together a pair with a possible value for the key by making partial encryptions and checking that the necessary conditions are fulfilled. We discard all the candidates that do not follow the characteristic.

We start by considering the ciphertext side and make a guess on the seven nibbles $K_0[1, 3, 4, 8, 9, 12, 16]$. We partially decrypt each of the pairs through the matching seven nibbles of the last Sbox layer, and look at the difference that we obtain: any candidate which difference is not one of the previously computed 16 possible values for ΔX_{18} is discarded.

We follow a similar procedure in plaintext side: we make a guess on the 28 key bits that intervene in the computation of the 7 active Sboxes ($(K_0 \oplus K_1)[4, 5, 6, 8, 9, 12, 16]$) and partially encrypt the corresponding nibbles. If the obtained difference is one of the 16 precomputed ones, we keep the candidate as possible, otherwise we discard it.

At this point, each pair is associated with $28 + 28 = 56$ bits of key corresponding to $(K_0 \oplus K_1)[4, 5, 6, 8, 9, 12, 16]$ and $K_0[1, 3, 4, 8, 9, 12, 16]$. From these possible values for parts of $(K_0 \oplus K_1)$ and of K_0 , we deduce possible values for $K_1[4, 8, 9, 12, 16]$. In addition to that, Property 1 implies that we can deduce nibble 4, 8 and 16 of any round key K_i .

We now have a look at the Sbox layer of round 2. We know the value of the differences entering Sbox 8 and 16, together with the value of $K_2[8, 16]$. To check if the Sboxes execute the right transitions, we lack the value of the two middle bits of nibble $I_2[8]$ and $I_2[16]$. By inverting the linear layer, we can see that these values depend on the values of 3 unknown bits which are bit 2, 42 and 59 of state Y_1 (see Fig. 3). The ANF description of the Sbox (see Sect. 6.1) indicates that the values of these 3 bits depend on 10 bits of the plaintext (which is known), together with 10 key bits: $(K_0 \oplus K_1)^{2,3,4}[1]$, $(K_0 \oplus K_1)^{2,3,4}[11]$ and $(K_0 \oplus K_1)[15]$ respectively. Consequently, the idea would be to make a guess on these 10 key bits, deduce the value of $X_2[8]$ and $X_2[16]$ and check whether the transitions are satisfied or not. The probability that a guess passes this test is 2^{-4} .

We follow a similar procedure to handle the last 2 rounds. Let us recall here that our 14-round characteristic ends at round 16 and that the difference spreads freely in rounds 17 and 18, which are respectively of type \mathcal{R} and \mathcal{R}' . Our goal here is to check the transitions of Sbox 8 and 16 of round 17 by using Property 2. To limit the complexity of this step, we only check that the value of x_2 is correct instead of checking both relations, so we are only interested in $Y_{17}^2[8]$ and $Y_{17}^2[16]$ (denoted c_2 and d_2 in Fig. 4). By referring to the linear layer, we obtain that their expressions in function of I_{18} are:

$$\begin{cases} Y_{17}^2[8] = I_{18}^2[6] \oplus I_{18}^2[7] \oplus I_{18}^2[14], \\ Y_{17}^2[16] = I_{18}^2[3] \oplus I_{18}^2[11] \oplus I_{18}^2[12]. \end{cases}$$

The value of I_{18} depends on ciphertext bits (state C) together with bits of K_0 and K_{18} . For instance, $Y_{17}^2[8]$ can be rewritten as:

$$\begin{aligned}
 Y_{17}^2[8] &= (K_{18}^2[6] \oplus X_{18}^2[6]) \oplus (K_{18}^2[7] \oplus X_{18}^2[7]) \oplus (K_{18}^2[14] \oplus X_{18}^2[14]) \\
 &= (K_{18}^2[6] \oplus K_{18}^2[7] \oplus K_{18}^2[14]) \oplus (Y_{18}^4[6] \oplus Y_{18}^3[6]Y_{18}^2[6]) \oplus (Y_{18}^4[7] \oplus Y_{18}^3[7]Y_{18}^2[7]) \\
 &\quad \oplus (Y_{18}^4[14] \oplus Y_{18}^3[14]Y_{18}^2[14]) \\
 &= (K_{18}^2[6] \oplus K_{18}^2[7] \oplus K_{18}^2[14]) \oplus ((C^4[6] \oplus K_0^4[6]) \oplus (C^3[6] \oplus K_0^3[6])(C^2[6] \oplus K_0^2[6])) \\
 &\quad \oplus ((C^4[7] \oplus K_0^4[7]) \oplus (C^3[7] \oplus K_0^3[7])(C^2[7] \oplus K_0^2[7])) \oplus ((C^4[14] \oplus K_0^4[14]) \\
 &\quad \oplus (C^3[14] \oplus K_0^3[14])(C^2[14] \oplus K_0^2[14])).
 \end{aligned}$$

Which indicates that we need to make a guess on:

$$K_0^{2,3}[6], K_0^{2,3}[7], K_0^{2,3}[14], K_{18}^2[6] \oplus K_{18}^2[7] \oplus K_{18}^2[14] \oplus K_0^4[6] \oplus K_0^4[7] \oplus K_0^4[14].$$

We follow a similar procedure for $Y_{17}^2[16]$ and conclude that we need to guess another 3 bits, namely:

$$K_0^{2,3}[11], K_0^4[11] \oplus K_{18}^2[3] \oplus K_{18}^2[11] \oplus K_{18}^2[12].$$

To sum up the key guessing process, we started from a set of $a \cdot 2^{84}$ possible pairs, we guessed $28 + 28 + 10 + 10 = 76$ key bits and we had access to a filter of $2^{-36} \cdot 2^{-24} \cdot 2^{-24} \cdot 2^{-4} \cdot 2^{-2} = 2^{-54}$ (which corresponds respectively to filtering on the ciphertext difference, checking last and first round and finally second and seventeenth rounds). The number of candidates remaining in the last step is then equal to $a \cdot 2^{70}$. So, in average, each of the 76-bit key candidate will be counted $a \cdot 2^{70} \cdot 2^{-76} = a \cdot 2^{-6}$ times, while as we expect to have a right pairs, the right key candidate will be counted a times. The *signal to noise ratio* (S/N) will then be equal to 2^6 , which ensure that we can distinguish the right key candidate from the wrong ones.

The last step of the attack consists in doing an exhaustive search to find the correct value for the remaining $128 - 76 = 52$ key bits.

6.3 Detailed Description of the Attack and of Its Complexities

In this section, we detail the time, data and memory complexities of our attack. We show that a naive implementation of the attack procedure described in Sect. 6.2 would lead to a time complexity overrun, and show how to deal with this issue.

Data Complexity: As detailed previously, we need about $a \cdot 2^{57}$ chosen plaintexts in order to successfully achieve the attack. We choose $a = 2^4$, which means that the data complexity of our attack is equal to 2^{61} . We recall that the security provided by PRIDE when the attacker has access to 2^d messages is equal to 2^{127-d} . Since our attack requires 2^{61} messages, we are limited to a number of operations lower than 2^{66} encryptions.

Time Complexity: To summarize the process described in Sect. 6.2, the attack is made of 6 main steps:

1. Encrypt 2^{33} structures and filter wrong pairs by looking at their input and output differences.
2. Make a guess on 28 bits of K_0 and check the transitions of the active Sboxes of last round. Eliminate wrong candidates, (that are associations of a pair with a key value that do not satisfy the transitions).
3. Make a guess on 28 bits of $K_0 \oplus K_1$ and check the transitions of the active Sboxes of first round. Eliminate wrong candidates.
4. Make additional guesses on 10 bits of $K_0 \oplus K_1$ to access the value entering Sbox number 8 and 16 of round 2 and check their transitions.
5. Make additional guesses on 10 bits of K_0 and K_{18} to access the value outputting Sbox number 8 and 16 of round 17 and check their transitions.
6. The key guess that is suggested the most is the correct one. Make a guess on remaining 52 key bits and do trial encryptions to recover the 128-bit master key.

A naive implementation of this process would lead to several problems. First, the attack involves many key bit guesses and uses many pairs, which would make the time complexity exceed our upper bound of 2^{66} PRIDE encryptions as soon as step 3. Second, detecting which key candidate is the most frequent would require to keep track of 2^{76} counters, which is clearly not reasonable.

As described next, we solve those two problems by making small guesses at the time and by studying each possible key guess for all possible pairs instead of studying each pair one after the other with all the possible key candidates.

First 3 Steps of the Attack. As briefly mentioned in Sect. 6.2, the first step of the attack consists in filtering the 2^{88} pairs of messages by looking at their plaintext and ciphertext differences.

Starting from the known 16 possible differences in ΔY_1 and ΔX_{18} , we refer to the difference distribution table and precompute the possible differences in P and C . A search returns that ΔP can take $170164 = 2^{17.38}$ values while ΔC can take $999448 = 2^{19.93}$ values. This implies that out of the 2^{88} initial pairs of messages only $2^{88} \cdot (2^{17.38} \cdot 2^{-28}) \cdot (2^{19.93} \cdot 2^{-64}) = 2^{33.31}$ pairs will remain.

In practice, we start by filtering pairs according to the truncated difference in the ciphertext. We are left with 2^{52} pairs whose differences on the plaintext and ciphertext sides are only on (at most) 7 nibbles. We then build two tables of 2^{28} bits each: the first table indicates if a difference on 28 bits is possible in the plaintext side (so contains a ‘1’ at the position corresponding to the $2^{17.38}$ possible ΔP), while the second indicates which 28-bit differences are valid on the ciphertext side. Each of the 2^{52} remaining pairs then requires at most two table look up to be filtered.

Then, we store all these $2^{33.31}$ pairs and evaluate them with all possible key values. This change implies that instead of needing 2^{76} counters, we have to save the $2^{33.31}$ pairs (so we require $2^{35.31}$ blocks of 64 bits).

In step 2, we start by guessing 7 nibbles of K_0 ($K_0[1, 3, 4, 8, 9, 12, 16]$). For each possible value we study the $2^{33.31}$ pairs, and cancel the ones that do not fulfill the required conditions. More precisely, the key guess is used to invert last round Sbox layer and compute the difference ΔX_{18} . We only keep pairs that lead to one of the 16 valid differences. Since there are 2^{28} possible values for the 7 key nibbles and that we repeat these operations for each of the $2^{33.31}$ pairs, this step is made $2^{61.31}$ times. To express this complexity in terms of PRIDE encryptions, we can see that it consists in computing two times 7 Sboxes (for a pair), while 1 full encryption with the cipher requires $18 \cdot 16 = 288 = 2^{8.17}$ Sbox computations. Step 2 is then roughly equivalent to $2^{56.95}$ PRIDE encryptions.

Step 3 consists in the same operations as step 2 but in plaintext side. We guess the 7 nibbles ($K_0 \oplus K_1$)[4, 5, 6, 8, 9, 12, 16] and compute the corresponding 7 Sboxes of round 1 for all the remaining pairs. The pairs that are processed in this step correspond to the pairs that remain after step 2, that is on average $2^{33.31} \cdot (16 \cdot 2^{-19.93}) = 2^{17.38}$ pairs associated to each possible value for the 28 bits of key. In its naive form, the number of PRIDE encryptions made in this step would then be equal to $2^{28} \cdot 2^{28} \cdot 2^{17.38} \cdot 7 \cdot 2 \cdot 2^{-8.17} = 2^{69.02}$, which exceeds our limit of 2^{66} PRIDE encryptions. To solve this problem, we encrypt one Sbox after the other and immediately check if the conditions are fulfilled. We start with Sbox number 5, for which the targeted output difference is 2. Given one of the 2^4 possible values for ($K_0 \oplus K_1$)[5] fixed, we compute $Y_1[5]$ (this requires a total of $2^{28} \cdot 2^4 \cdot 2^{17.38} \cdot 2 = 2^{50.38}$ Sbox operations) and check that the obtained difference is equal to 2. To compute the number of pairs that pass this test we need to take into account the proportion of pairs for which Sbox number 5 is active (equal to $\frac{152004}{170164} = 2^{-0.16}$) together with the probability that an active Sbox of our pre-filtered set leads to a difference of 2 (which is $\frac{1}{6}$). We obtain the following estimate:

$$2^{17.38} \cdot \left(\frac{152004}{170164} \cdot \frac{1}{6} + \frac{18160}{170164} \cdot 1 \right) = 2^{17.38} \cdot 2^{-1.97} = 2^{15.41}.$$

In the same way, we make a guess on the 4 bits of ($K_0 \oplus K_1$)[6], which requires $2^{28} \cdot 2^4 \cdot 2^4 \cdot 2^{15.41} \cdot 2 = 2^{52.41}$ Sbox encryptions. We then filter out wrong pairs by checking that active Sboxes give a difference of 2. The number of remaining pairs is then equal to⁸ $2^{13.37}$. We then process Sbox number 9, which requires $2^{28} \cdot (2^4)^3 \cdot 2^{13.37} \cdot 2 = 2^{54.37}$ Sbox encryptions and leaves us with an average of $2^{12.33}$ pairs for each partial key guess. Next, we treat Sbox number 4 and 12, taking advantage of the fact that they must have the same output difference. The number of Sbox encryptions is equal to $2^{61.33}$ and $2^{8.73}$ pairs remain in average. We finally handle the last 2 Sboxes together, which requires $2^{65.73}$ Sbox encryptions and discard all but 2^4 pairs in average for each key candidate⁹. To sum up, total time complexity of this step is $2^{50.38} + 2^{52.41} + 2^{54.37} + 2^{61.33} + 2^{65.73} = 2^{65.80}$ which is equivalent to $2^{66.80-8.17} = 2^{57.63}$ PRIDE encryptions.

⁸ The computation of the quantities used in this step are detailed in the full version of the paper [10].

⁹ $2^{17.38} \cdot (16 \cdot 2^{-17.38}) = 2^4$.

Steps 4 and 5. Next operations (step 4 and 5) consist in checking the transitions of Sbox number 8 and 16 in round 2 and in round 17. As explained before, we look at the value of only 3 out of their 4 input bits in round 2 and only 1 out of 4 output bits in round 17. In the following, we name these bits a_i, b_i , ($2 \leq i \leq 4$) and c_2, d_2 , respectively (see Figs. 3 and 4).

We start by explaining how to check the 2 active Sboxes of round 2. We remark here that if a_2 (respectively b_2) defines a condition on its own, the condition that a_4 (resp. b_4) must fulfill sometimes depends on a_3 (resp. b_3).

As briefly mentioned in Sect. 6.2 and as illustrated in Fig. 3, a_2 and b_2 are given by the following two expressions:

$$\begin{aligned} a_2 &= K_2^2[8] \oplus Y_1^2[1] \oplus Y_1^2[8] \oplus Y_1^2[11] \\ b_2 &= K_2^2[16] \oplus Y_1^2[8] \oplus Y_1^2[11] \oplus Y_1^2[12] \end{aligned}$$

that when referring to Definition 1 can be rewritten as:

$$\begin{aligned} a_2 &= K_2^2[8] \oplus P^4[1] \oplus (K_0 \oplus K_1)^4[1] \\ &\quad \oplus (P^3[1] \oplus (K_0 \oplus K_1)^3[1]) \cdot (P^2[1] \oplus (K_0 \oplus K_1)^2[1]) \\ &\quad \oplus Y_1^2[8] \oplus P^4[11] \oplus (K_0 \oplus K_1)^4[11] \\ &\quad \oplus (P^3[11] \oplus (K_0 \oplus K_1)^3[11]) \cdot (P^2[11] \oplus (K_0 \oplus K_1)^2[11]) \quad (1) \\ b_2 &= K_2^2[16] \oplus Y_1^2[8] \oplus P^4[11] \oplus (K_0 \oplus K_1)^4[11] \\ &\quad \oplus (P^3[11] \oplus (K_0 \oplus K_1)^3[11]) \cdot (P^2[11] \oplus (K_0 \oplus K_1)^2[11]) \\ &\quad \oplus Y_1^2[12], \quad (2) \end{aligned}$$

for which the only unknown bits are $(K_0 \oplus K_1)^{\{2,3,4\}}[1]$ and $(K_0 \oplus K_1)^{\{2,3,4\}}[11]$. Indeed, the plaintext bits are known and $K_2^2[8]$ and $K_2^2[16]$ are deduced from key-schedule properties, while a_2 and b_2 are determined by the difference observed in X_2 together with the relations given by Property 2.

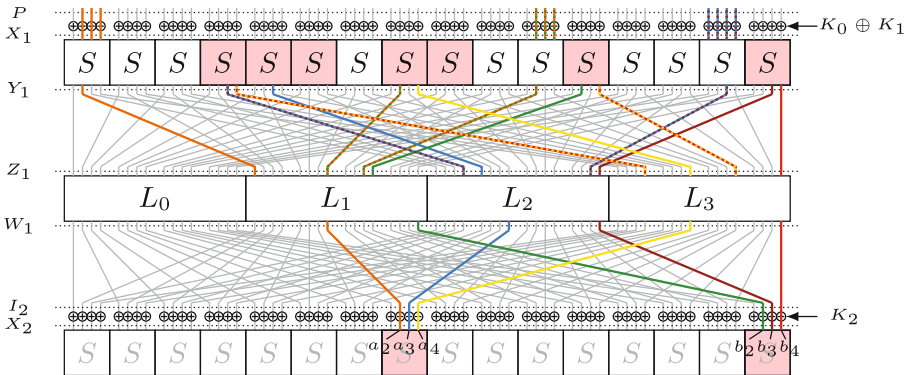


Fig. 3. Bits involved in the computation of a_2, a_3, a_4 and b_2, b_3, b_4 .

Consequently, we make a guess on these 6 key bits and check that the relations given by Eqs. (1) and (2) hold, which happens with probability 2^{-2} .

Since we have an average of 2^4 candidates for each possibility for the 56 bits of key guessed so far, this step is repeated $2^4 \times 2^{56} \times 2^6 = 2^{66}$ times. We expect that $2^4 \times 2^{56} \times 2^6 \times 2^{-2} = 2^{64}$ candidates remain after it. Since computing a_2 and b_2 requires less operations than for an Sbox encryption, the time complexity of this step is less than $2 \cdot 2^{66-8.17} = 2^{58.83}$ full cipher encryptions.

We then look at a_3 , a_4 , b_3 and b_4 . As can be seen in Fig. 3, the bits that are necessary to compute a_4 and b_4 are $Y_1^4[4]$, $Y_1^4[8]$, $Y_1^4[12]$, $Y_1^4[16]$, $K_2^4[8]$ and $K_2^4[16]$. Since all these bits are known from previous computations, we can obtain a_4 and b_4 and deduce from the value of ΔY_1 and Property 2 the conditions that they must fulfill on their own or with respect to a_3 and b_3 .

To simplify the explanation, we consider that a_3 and b_3 are always necessary to check the Sboxes. Note that this simplification is at the disadvantage of the attacker and results in an over estimation of the time complexity.

Bits a_3 and b_3 are given by the following expressions (see also Fig. 3):

$$\begin{aligned} a_3 &= K_2^3[8] \oplus Y_1^3[4] \oplus Y_1^3[5] \oplus Y_1^3[15] \\ b_3 &= K_2^3[16] \oplus Y_1^3[4] \oplus Y_1^3[15] \oplus Y_1^3[16] \end{aligned}$$

in which the only unknown bit is $Y_1^3[15]$. Since this term appears linearly in both a_3 and b_3 , we can obtain a relation relying only on known bits by xoring the two expressions:

$$a_3 \oplus b_3 = K_2^3[8] \oplus K_2^3[16] \oplus Y_1^3[5] \oplus Y_1^3[16].$$

Therefore without any key guessing we can filter our candidates and reduce their number by a factor of 2^{-1} : as a result, 2^{63} candidates remain at this point, while the complexity of this step is lower than $2^{64-8.17} = 2^{55.83}$ encryptions.

For the remaining candidates, we guess $(K_0 \oplus K_1)[15]$ to be able to compute $Y_1^3[15]$ and we check that a_3 takes the right value. This requires a guess of 4 bits, and leads to a reduction of the set of possible candidates by a factor of 2^{-1} . With 2^{67} simple computations (each roughly equal to one Sbox computation, so with a time complexity that is less than $2^{58.83}$ PRIDE encryptions), we reduce the number of candidates to 2^{66} .

At this point, we have 2^{66} candidates made of a pair of plaintext/ciphertext associated to a guessed value for 66 key bits. The average count for a wrong key is expected to be 1, while we built our messages so that the right key appears around $a = 2^4$ times.

The distribution of keys in the candidates follows a binomial distribution of parameters $n = 2^{66}$ and $p = 2^{-66}$ ($B(2^{66}, 2^{-66})$) that can be approximated by a Poisson distribution of parameter $\lambda = np = 1$ so the probability that a wrong key appears strictly more than t times in our set of candidates is given by:

$$P_t = 1 - \sum_{k=0}^t e^{-1} \cdot \frac{1^k}{k!} = 1 - e^{-1} \cdot \sum_{k=0}^t \frac{1}{k!}$$

The idea here is to do an additional filtering step (that is checking the 2 active Sboxes of round 17) only for candidates that are associated with a key that is suggested $t + 1$ times or more. Doing so, the ratio of candidates that we have to study is equal to P_t .

We choose $t = 13$, meaning that we are now looking at $2^{66} \cdot 2^{-37.7} = 2^{28.3}$ candidates. For these, we start by computing the value of c_2 and d_2 , that as can be seen in Fig. 4 depend on the following unknown bits:

- For c_2 : $K_0^{\{2,3\}}[6]$, $K_0^{\{2,3\}}[7]$, $K_0^{\{2,3\}}[14]$ and $K_{18}^2[6] \oplus K_{18}^2[7] \oplus K_{18}^2[14] \oplus K_0^4[6] \oplus K_0^4[7] \oplus K_0^4[14]$.
- For d_2 : $K_0^{\{2,3\}}[11]$ and $K_0^4[11] \oplus K_{18}^2[3] \oplus K_{18}^2[11] \oplus K_{18}^2[12]$.

We start by guessing the 3 key bits required to compute d_2 , and we filter our guesses by confronting the obtained value with the value given by Property 2. The filtering ratio is of 2^{-1} , so the number of candidates after this step is: $2^{28.3} \times 2^3 \times 2^{-1} = 2^{30.3}$.

Next, we repeat the same process by guessing the 7 unknown key bits that are necessary to compute c_2 . The number of candidates obtained at this point is: $2^{30.3} \times 2^7 \times 2^{-1} = 2^{36.3}$, and the time complexity of these two steps is negligible in comparison to previous ones.

For all the key candidates that are (still) suggested 14 times or more, we do an exhaustive search to find the value of the $128 - 76 = 52$ unknown key bits and check them by doing a trial encryption. Since we expect $2^{11.3}$ such key candidates, this step will at most require $2^{11.3} \cdot 2^{52} = 2^{63.3}$ encryptions.

To sum up, the total data complexity of our attack is 2^{61} chosen plaintexts, its time complexity is less than $2^{63.3}$ 18-round PRIDE encryption and its memory complexity is of 2^{35} 64-bit blocks.

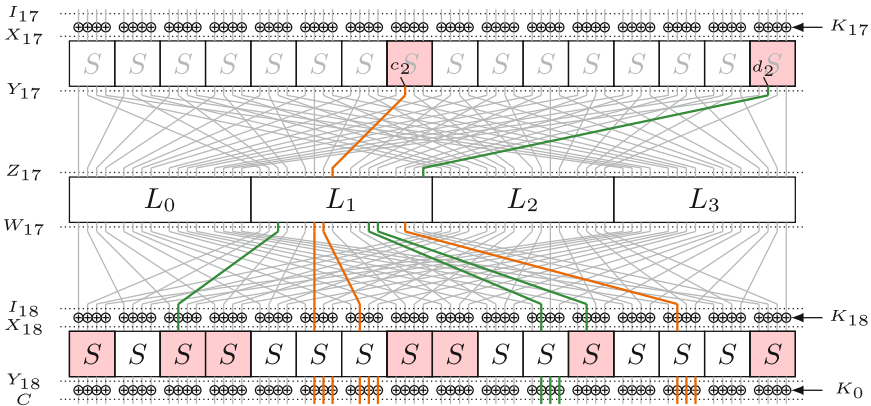


Fig. 4. Bits involved in the computation of c_2 and d_2 .

7 Conclusion




In this paper, we studied the resistance of PRIDE against differential cryptanalysis. We first proved that two previous differential attacks are wrong since essential bits are unknown to the attacker, making her unable to succeed. Our main contribution is a 18-round differential cryptanalysis of the cipher that results from a careful analysis of its high probability characteristics and of its diffusion layer. Our attack recovers the full 128-bit master key with 2^{61} chosen plaintexts, a time complexity equivalent to $2^{63.3}$ encryptions and requires to store around 2^{35} 64-bit blocks.

References

1. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçın, T.: Block ciphers – focus on the linear layer (feat. PRIDE). In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 57–76. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_4
2. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK lightweight block ciphers. In: Proceedings of the 52nd Annual Design Automation Conference, 2015, pp. 175:1–175:6. ACM (2015)
3. Dai, Y., Chen, S.: Cryptanalysis of full PRIDE block cipher. *Sci. China Inf. Sci.* **60**(5), 052108:1–052108:12 (2017)
4. Dinur, I.: Cryptanalytic time-memory-data tradeoffs for FX-constructions with applications to PRINCE and PRIDE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 231–253. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_10
5. 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
6. Guo, J., Jean, J., Mouha, N., Nikolic, I.: More Rounds, Less Security? IACR Cryptology ePrint Archive 2015, 484 (2015)
7. Karakoç, F., Demirci, H., Harmanç, A.E.: ITUbee: a software oriented lightweight block cipher. In: Avoine, G., Kara, O. (eds.) LightSec 2013. LNCS, vol. 8162, pp. 16–27. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40392-7_2
8. Kilian, J., Rogaway, P.: How to protect DES against exhaustive key search (an analysis of DESX). *J. Cryptol.* **14**(1), 17–35 (2001)
9. Lac, B., Beunardeau, M., Canteaut, A., Fournier, J.J., Sirdey, R.: A First DFA on PRIDE: from Theory to Practice (extended version). IACR Cryptology ePrint Archive 2017, 075 (2017)
10. Lallemand, V., Rasoolzadeh, S.: Differential cryptanalysis of 18-round PRIDE. IACR Cryptology ePrint Archive 2017, 1017 (2017)
11. Standaert, F.-X., Piret, G., Gershenfeld, N., Quisquater, J.-J.: SEA: a scalable encryption algorithm for small embedded applications. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 222–236. Springer, Heidelberg (2006). https://doi.org/10.1007/11733447_16
12. Tezcan, C.: Improbable differential attacks on present using undisturbed bits. *J. Comput. Appl. Math.* **259**, 503–511 (2014)

13. Tezcan, C., Okan, G.O., Şenol, A., Doğan, E., Yücebaş, F., Baykal, N.: Differential attacks on lightweight block ciphers PRESENT, PRIDE, and RECTANGLE revisited. In: Bogdanov, A. (ed.) LightSec 2016. LNCS, vol. 10098, pp. 18–32. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55714-4_2
14. Tezcan, C., Özbudak, F.: Differential factors: improved attacks on SERPENT. In: Eisenbarth, T., Öztürk, E. (eds.) LightSec 2014. LNCS, vol. 8898, pp. 69–84. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16363-5_5
15. Yang, Q., Hu, L., Sun, S., Qiao, K., Song, L., Shan, J., Ma, X.: Improved differential analysis of block cipher PRIDE. In: Lopez, J., Wu, Y. (eds.) ISPEC 2015. LNCS, vol. 9065, pp. 209–219. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17533-1_15
16. Zhao, J., Wang, X., Wang, M., Dong, X.: Differential Analysis on Block Cipher PRIDE. IACR Cryptology ePrint Archive 2014, 525 (2014)

DSA Signing Key Recovery with Noisy Side Channels and Variable Error Rates

Jiji Angel^(✉), R. Rahul, C. Ashokkumar, and Bernard Menezes

Indian Institute of Technology Bombay, Mumbai, India
{jiji,rahul,ashokkumar,bernard}@cse.iitb.ac.in

Abstract. The Digital Signature Algorithm (DSA) computes a modular exponentiation with a per-message ephemeral secret. This involves a sequence of modulo square and multiply operations which, if known, enables an adversary to obtain the DSA private key. Cache-based side channel attacks are able to recover only discontinuous blocks of the ephemeral key thanks to the sliding window optimization implemented in many crypto libraries. Further, noisy side channels, rarely addressed in the literature, greatly complicate key retrieval. Through extensive experiments, we obtain estimates of the error rate as a function of block position and size. We demonstrate key retrieval in the presence of noise and model the time complexity of key recovery as a function of error rate. Our model exposes the tradeoff between number of signature operations that need to be monitored and the computational requirements for the attack. By selectively using interior blocks in the ephemeral key, we are able to retrieve the DSA private key with less than half the number of signatures required by previous work that use only the rightmost block.

Keywords: Cryptanalysis · Lattice attack · DSA

1 Introduction

The Digital Signature Algorithm (DSA) [1] and its elliptic curve variant, ECDSA [1] belong to the ElGamal family of signature schemes. Proposed by NIST and adopted as FIPS 186, these algorithms are part of Digital Signature Standard (DSS). The security of these algorithms is based on the presumed intractability of the Discrete Logarithm Problem (DLP). However, they are known to be vulnerable to side channel attacks - these include attacks which exploit features of their software implementations.

There has been much work on cache-based side channel attacks on various cryptographic algorithms including AES [2], RSA [3, 4], DSA [5, 6] and EC-DSA [6–9]. They exploit the fact that access times to main memory are about an order of magnitude greater than access times to processor cache. Though there are many variations of cache access attacks, the victim (performing DSA signature operations) and the attacker or spy typically reside on the same physical machine. In some attacks, the victim and spy are assumed to be on the same core [2, 10].

However, when two cores share the same lower level cache (L3 cache) as is the case with many modern processors, it is still possible to implement some cache attacks [11]. The attacks implemented in this paper use the second scenario. We target the DSA implementation in the cryptographic library of OpenSSL version 0.9.8.

A DSA signature operation involves computing a modular exponentiation where the exponent is a randomly selected per-message secret referred to as the ephemeral key. Knowing an ephemeral key together with its associated message and signature enables an attacker to compute the DSA signing key and thus forge signatures. The standard way of implementing modular exponentiation is through a series of square and multiply operations - both operations are performed modulo a large (160 bit or 256 bit) prime. Moreover, most software implementations of DSA employ Sliding Window Exponentiation to speed up modular exponentiation.

Using carefully timed accesses to the cache, the spy will be able to record the SM sequence (squares and multiplies performed by the victim). Even if the exact SM sequence is obtained, the spy will be able to deduce only some (non-contiguous) blocks of the ephemeral key, not the complete key. Such blocks, one or more per signature computation, serve as input to well-researched lattice problems such as the Shortest Vector Problem (SVP) or Closest Vector Problem (CVP). Solutions to these problems yield the DSA signing key [5–8, 12].

Most previous efforts in DSA key retrieval assume a perfect (noise-free) side channel. However, despite efforts at improving the accuracy of the side channel information, our experimental results indicate error rates of over 25% based on the strict definition of the recovery of the entire SM sequence per signature. The focus of this paper is to obtain the DSA signing key even in the presence of such a noisy side channel.

Our main contributions can be summed up as follows. We first outline a strategy to retrieve the DSA private key with as few as 9^1 signatures in a perfect side channel. We then conduct experiments to study the error rate in recovering the partial ephemeral key through cache-based side channels in a contemporary processor where the victim and spy are hosted on separate cores. While [12] use the least significant bits of the ephemeral key as input to a hard lattice problem to recover the DSA private key, we also use interior blocks of leaked bits to reduce the number of required signatures to less than half. In addition to implementing a strategy to recover the DSA key, we present a model to predict the number of signatures and computation time required as a function of error rate.

The paper is organized as follows. In Sect. 2, we present background information. Section 3 describes our experiments in collecting the SM sequences and estimating the error rates involved. Section 4 outlines a simple approach for key retrieval assuming, both, a perfect and a noisy side channel. In Sect. 5, we present our analytical model and results. Section 6 briefly presents related work and Sect. 7 summarizes and concludes the paper.

¹ Typically an attacker obtains a large number of signatures. However only a fraction of these may actually be used in computing the DSA key.

2 Preliminaries

2.1 Cache Memories

All modern processors have multiple levels of cache intended to bridge the latency gap between main memory and the CPU. One of the target machines in this work is the Intel Core i7 which has three levels of cache. At the highest level is the L1 instruction (32 KB) and L1 data cache (32 KB). Each of its four cores has a 256 KB L2 cache. The four cores share a 3 MB L3 cache.

The granularity of cache access is a block or line which is 64 bytes in our target machines. The lines of a cache are organized into sets - a line from main memory can be placed in exactly one set though it may occupy any position in that set. The number of lines in a set is the associativity of the cache. The L1 and L2 caches in core i3 and i5 are 8-way set associative while L3 is 12-way set associative.

The Flush and Reload attack strategy [11] implemented in this paper depends on the inclusive property of L3 cache. Simply put, L3 is inclusive if it contains all lines found in L1 and in L2. So, flushing (evicting) a line in L3 guarantees that it will not be present either in L1 or L2.

2.2 Digital Signature Algorithm

The DSA standard [1] documents the algorithm under the following components: *Selection of Domain Parameters*, *Domain Parameter Generation*, *DSA Key Pair Generation*, *DSA Signature Generation* and *DSA Signature Verification and Validation*.

Domain Parameter Generation: Primes p, q such that $q|(p-1)$. g is a generator of a subgroup of order q in the multiplicative group of $GF(p)$. The sizes of the p and q used in this paper are 1024 bits and 160 bits respectively.

DSA Key Pair Generation: The per user keys for performing the digital signature operation are as follows. The private key α , is a randomly chosen integer from the range $[1, q-1]$. The public key is $y = g^\alpha \pmod{p}$. The modular exponentiation, $g^\alpha \pmod{p}$ is performed using efficient algorithms such as exponentiation by squaring but computing α from y is considered intractable (Discrete Logarithm Problem). This ensures the security of the private key α .

DSA Signature Generation: The following steps are carried out in signature generation:

- (Pseudo) Randomly choose integer k , $0 < k < q$. k is referred to as an ephemeral key and is per-signature unique.
- Compute: $r = (g^k \pmod{p}) \pmod{q}$ and

$$s = (k^{-1}(h(m)) + \alpha r) \pmod{q} \quad (1)$$

where k^{-1} is the multiplicative inverse of k modulo q and $h(m)$ is the hash of the message m to be signed. The hash function h is an approved secure hash function such as SHA-256 or SHA-512.

In the extremely unlikely case when $r = 0$ or $s = 0$, proceed with a different value of k .

- The pair (r, s) is the signature and is sent along with the message m .

DSA Signature Verification and Validation: The signature is verified as follows.

- Let m', r', s' be the received versions of m, r, s respectively.
- If $r' = 0$ or $s' = 0$, the signature is considered invalid and is rejected.
- Compute:
 - ◊ $u_1 = (h(m')(s')^{-1}) \bmod q$
 - ◊ $u_2 = r'(s')^{-1} \bmod q$
 - ◊ $v = ((g^{u_1}y^{u_2}) \bmod p) \bmod q$
- The signature pair is authentic, if $v = r'$.

3 Side Channel Attacks

3.1 Background

Left-to-right computation of the modular exponentiation, $g^e \bmod p$ consists of a series of squares and multiplies. Starting from the MSB and with $temp = 1$, for each bit scanned, a squaring is performed, i.e., $temp \leftarrow temp^2 \pmod p$. When a 1 is encountered, a multiplication by g is also performed (in addition to the squaring), i.e. $temp \leftarrow (temp * g) \pmod p$. On average, it could be assumed that half the bits in the ephemeral key are 1's, so the number of multiplications is roughly half the bit length of e .

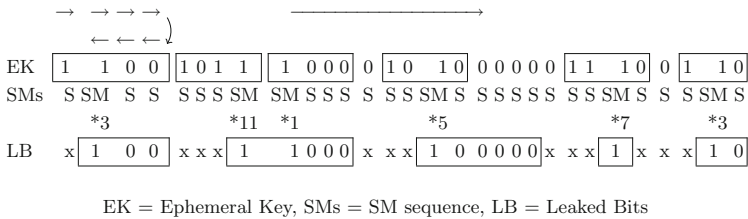


Fig. 1. Illustrating leaked blocks from an SM sequence

To reduce the number of multiplies, sliding window exponentiation is employed. Here, a window of w consecutive bits is made to slide down the ephemeral key, left to right. If the MSB of the window is 1, the bits within the window are scanned leftward from its LSB end until a 1 is encountered. This is the point at which a multiply is to be performed, i.e. $temp \leftarrow (temp * x) \pmod p$ where x is the value whose binary representation is the bit string from the MSB of the window to this point. The window is made to slide by w or more bits to the right until a window with MSB = 1 is found. Figure 1 illustrates the sequence of operations using a toy 31-bit exponent. The first row shows the ephemeral key while the second row is the corresponding SM sequence obtained from Sliding

Window Exponentiation with $w = 4$. The third row shows the integer to be multiplied by *temp*. The sliding window technique reduces the number of multiplies by roughly a factor of $(w + 1)/2$. In OpenSSL version 0.9.8, $w = 4$ is used.

The value of x to be multiplied by *temp* is always g^y , where y is an odd integer ranging between 1 and $2^w - 1$. Thus, the values g^3, g^5, \dots must be pre-computed and stored. The reduced number of multiplications during the computation of the modular exponentiation, however, far outweighs the cost of pre-computation.

The fourth row shows the bits of the ephemeral key that are deduced by inspection of the SM sequence. There are 5 blocks of known bits. Other than the rightmost block, we refer to all other blocks (including the leftmost) as interior blocks. The exact sequence of bits in a block together with its exact position is a possible input to the lattice problem (Sect. 4).

3.2 Retrieving the SM Sequence

Our experiments were conducted on an Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz with 4 GB memory running Linux kernel 3.16.35. Each core has its private L1 (8-way 32 KB data and 32 KB instruction caches) and L2 caches (8-way unified 256 KB). The last level cache (LLC) is a 12-way, 8 MB unified cache and is shared between all cores. The line size in all cases is 64 bytes. Both the spy and victim are assumed to be on the same machine but on different cores. The victim is computing DSA signatures using the Open SSL cryptographic library version 0.9.8. The OpenSSL code is mapped to the virtual spaces of the victim and spy, so both can access the code. While the victim is computing a modular exponentiation with the ephemeral key as exponent, the spy is continually performing a Flush + Reload attack [10, 11].

The code for the modulo multiply and square operations occupy 15 and 11 lines of cache respectively. Each probe by the spy involves accessing a total of 8 cache lines, four each in the multiply and square code. These lines include one each from the frequently invoked `bn_mul_recursive()` and `bn_sqr_recursive()` functions. After accessing these lines, the spy flushes them from all levels of the cache hierarchy using the `clflush` instruction. If any of these lines are accessed by the victim between two probes, they would have been brought into L3 cache from main memory. So, the spy will be able to determine which line(s) were accessed by the victim since the access time to main memory is substantially higher than that to L3 cache. The spy concludes that a multiply (respectively square) operation has been executed if even a single line in the multiply (respectively square) code has been found to be accessed by the victim.

The multiply and square operations take roughly 2438 and 2192 nsec. During a single multiply, the spy probes the cache either twice (63% of the time) or 3 times (30%). The corresponding numbers for the square operation are once (25%) or twice (72%). Two or more square operations can occur consecutively (this is not the case with the multiply). Since the number of probes in a square varies, it is possible to misjudge the number of consecutive squares - and hence zeros in the ephemeral key. Fortunately, many square/multiply operations are followed by probes that detect no activity on any of the cached lines.

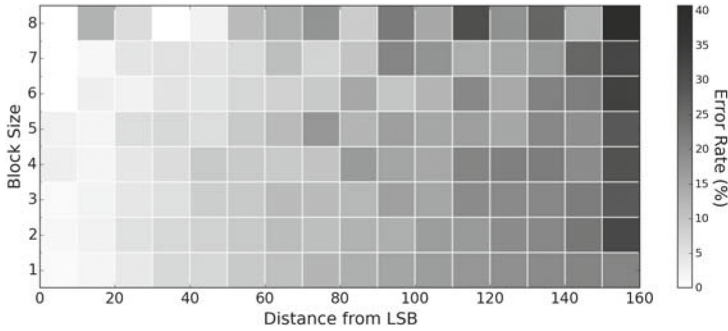


Fig. 2. Heat-map showing error rate of blocks

We extracted the SM sequences corresponding to the ephemeral keys used in several thousand DSA signing operations. Of these roughly 3% were clearly erroneous - for example, the length of the sequence was too long or too short. Of the remainder, 75% matched the actual SM sequence perfectly. From the SM sequence, we reconstructed the partial ephemeral key. A block is in error if its value and/or position as deduced from the SM sequence differs from that in the ephemeral key. In estimating the average block error probability, we considered only large interior blocks (of minimum length 5) and rightmost blocks and excluded all others since these are the only blocks of interest to the key retrieval algorithm. Figure 2 shows a heat map of the block error probability as a function of its distance from the LSB and its size. The average error rate is only about 1% for rightmost blocks, increases to about 10–15% in the mid section and reaches a high of around 40% in the extreme left. Hence, we discarded blocks that lie within 40 bits of the leftmost end of the ephemeral key.

4 DSA Private Key Retrieval

Retrieval of the DSA signing key is formulated as a hard and well-known lattice problem [5]. The known blocks of the ephemeral key are inputs to this problem. First, the lattice problem is presented. Then, the key retrieval algorithms which create and solve instances of the hard lattice problem are presented.

4.1 Lattice Basics

A full-rank n -dimensional lattice is a set $L = \mathbb{Z}\mathbf{b}_1 + \dots + \mathbb{Z}\mathbf{b}_n$, i.e., L is the space of all integer linear combinations of some linearly independent vectors \mathbf{b}_i 's in \mathbb{R}^n . The \mathbf{b}_i 's are a basis of \mathbb{R}^n and form a basis of L . The length of a lattice vector, $x \in L$ is computed as the Euclidean norm of the vector and is denoted $\|x\|$. Among many of the lattice-based NP-hard problems, the following are of importance in the cryptanalysis of digital signature algorithms such as RSA [3, 4], DSA [5, 6] and ECDSA [7, 8, 13].

Table 1. Symbols used in key recovery algorithms and performance analysis

Symbol	Meaning	Value
n	No. of signatures	100, 140
T	Set of top-ranked EKO's	-
τ	Cardinality of T	70
A	Set of EKO's actually used in an instance of SVP	-
γ	No. of EKO's used in an instance of SVP in Algorithm 3	30 for $n = 140$, 34 for $n = 100$
γ_{min}	Minimum no. of EKO's used in an instance of SVP in Algorithm 2	20
γ_{max}	Maximum no. of EKO's used in an instance of SVP in Algorithm 2	50 for $n = 140$, 60 for $n = 100$
ρ	No. of randomized shufflings of EKO's performed in each outer loop iteration of Algorithm 3	50 for $n = 140$, 100 for $n = 100$
$p_{i,n}$	Probability of success in an iteration of the inner loop of Algorithm 2 involving i error-free EKO's	-
p_i	Probability that all the i EKO's participating in an iteration of the inner loop of Algorithm 2 are error-free	-
P_s	Success probability of an attempt (iteration of the outer loop)	-
ϑ	Number of attempts (iterations of the outer loop) required to achieve success	-

Shortest Vector Problem (SVP): Given the lattice L , the *Shortest Vector Problem* is to find a non-zero vector $x \in L$ such that $\|x\| \leq \|y\|, \forall y \in L$.

Closest Vector Problem (CVP): Given the lattice L and a non-lattice vector $t \in \mathbb{R}^n$, the *Closest Vector Problem* aims to compute a lattice vector $x \in L$ such that $\|x - t\| = \min_{y \in L} \|y - t\|$. There are two methods of solving the *CVP*. The first is to use exact *CVP* solvers such as Babai's algorithms [14]. The second approach is called the *embedding technique*, in which the *CVP* instance is reduced to an *SVP* instance which is then solved using a *SVP* solver such as LLL [15] or BKZ [16, 17]. The embedding technique transforms an n dimensional lattice basis M into a new basis M' which is of dimension $n + 1$. M' is defined as

$$M' = \begin{pmatrix} M & \mathbf{0} \\ t & q \end{pmatrix} \quad (2)$$

4.2 Lattice Attack Using CVP/SVP

We assume that we have access to a total of n message, signature pairs $(m_i, (r_i, s_i))$, and the SM sequences corresponding to each of the ephemeral

keys k_i (obtained from the side channel attack). From the SM sequences, we derive the partial ephemeral keys. We focus on the case where two substrings of bits in each ephemeral key are known - one or more consecutive rightmost bits and an “interior” substring flanked by two unknown substrings. Let a_i and c_i respectively denote the values of the two known substrings and d_i (on the left) and b_i (on the right) denote the two substrings that flank c_i . Also let $l_{a,i}$, $l_{b,i}$, $l_{c,i}$ denote the bit positions of the rightmost edges of b_i , c_i and d_i respectively. So,

$$k_i = d_i 2^{l_{c,i}} + c_i 2^{l_{b,i}} + b_i 2^{l_{a,i}} + a_i, \quad 0 \leq i < n \quad (3)$$

Substituting the value of k_0 from (3) into (1) and rearranging terms yields an expression for the private key α . This expression is substituted into (1), which together with (3) for $i = 1, 2, \dots, n - 1$ yields

$$b_i = w_{1,i} b_0 + w_{2,i} d_0 + u_i d_i - v_i + h_i q, \quad 1 \leq i < n \quad (4)$$

Let $Y_i = \frac{-r_i}{s_i} 2^{l_{a,i}}$ and $X_i = \left\{ \frac{-h(m_i)}{c_i} + c_i 2^{l_{b,i}} + a_i \right\} 2^{l_{a,i}}$. So, $w_{1,i} = \frac{Y_i}{Y_0}$, $w_{2,i} = \frac{Y_i}{Y_0} 2^{l_{c,i} - l_{a,i}}$, $u_i = -2^{l_{c,i} - l_{a,i}}$ and $v_i = X_i - \left(\frac{Y_i}{Y_0} \right) X_0$. From (4) we construct the basis matrix, M , for the lattice $L(M)$.

$$M_{2n \times 2n} = \left[\begin{array}{c|cccc} & w_{1,1} & w_{1,2} & \dots & w_{1,n-1} \\ & w_{2,1} & w_{2,2} & \dots & w_{2,n-1} \\ I_{n+1,n+1} & u_1 & 0 & \dots & 0 \\ & 0 & u_2 & \dots & 0 \\ & \vdots & & \ddots & \vdots \\ & 0 & 0 & \dots & u_{n-1} \\ \hline O_{n-1} & & & & qI_{n-1} \end{array} \right]$$

In the lattice $L(M)$, there exists a lattice vector

$$\begin{aligned} \mathbf{w} &= (b_0, d_0, d_1, \dots, d_{n-1}, h_1, \dots, h_{n-1}) \times M \\ &= (b_0, d_0, d_1, \dots, d_{n-1}, b_0 w_{1,1} + d_0 w_{2,1} + d_1 u_1 + h_1 q, \dots, \\ &\quad b_0 w_{1,n-1} + d_0 w_{2,n-1} + d_{n-1} u_{n-1} + h_{n-1} q) \end{aligned} \quad (5)$$

Let $\mathbf{b} = (b_0, d_0, d_1, \dots, d_{n-1}, d_1, \dots, d_{n-1})$ be the vector to be computed where the pair $\langle d_i, b_i \rangle$ constitute the unknown blocks of the i^{th} ephemeral key. Thus we have $\mathbf{w} - \mathbf{v} = \mathbf{b}$ where $\mathbf{v} = (0, 0, \dots, 0, v_1, v_2, \dots, v_{n-1})$ is the target (non-lattice) vector. The *Closest Vector Problem* is to find the lattice vector \mathbf{w} . This would retrieve the ephemeral keys and, in turn, the private key.

In general, the positions and lengths of known blocks will be different across keys. To take into account the variation in the lengths of, both, the rightmost and interior known blocks, across ephemeral keys, we multiply the basis matrix, M , using a weight matrix as in [5]. However, our weight matrix elements are much smaller. We construct the weight matrix as the diagonal matrix: $D_{(2n \times 2n)} = [d_{i,i}]$, where $d_{i,i}$ is computed based on the difference in block lengths between the largest block and the i^{th} block.

$$d_{i,i} = 2^{\max_block_size - \max\{len(b_i), len(d_i)\}} \quad (6)$$

where $\max_block_size = \max_{0 \leq i < n} \max\{len(b_i), len(d_i)\}$.

The CVP instance is converted into an SVP problem using an *embedding technique*. The new basis matrix M of dimension $2n + 1$ is $M' = \begin{pmatrix} M & 0 \\ \mathbf{v} & q \end{pmatrix} \times D$.

4.3 Retrieving the DSA Key from a Noise-Free Side Channel

For each ephemeral key, including an interior block as input to the SVP instance necessitates an increase by 2 in the number of lattice dimensions but including only the rightmost block increases the number of dimensions by only 1. Since the complexity of SVP increases greatly with the number of dimensions, including the rightmost block of each ephemeral key may be an attractive option. However, larger blocks leak more bits of the key and the largest interior block is almost always larger than the rightmost block. This suggests combining a large interior block with the rightmost block.

Algorithm 1. Key Retrieval using a noise-free side channel

Input: n triplets of <message, signature, SM sequence>
Output: DSA Private Key

- 1 $S = \{\}$ // S is a sorted list of EKO's
- 2 $keyNotFound = true$
- 3 **foreach** *signature* **do**
- 4 Identify all the large interior blocks and the rightmost block from the SM sequence
- 5 **foreach** *such block* **do**
- 6 Create an EKO, e , using the position and value of the interior block and the position and value of the rightmost block
- 7 $S = S \cup \{e\}$
- 8 **end**
- 9 **end**
- 10 Sort S on the basis of the sum of the lengths of the interior block and the rightmost block
- 11 Let γ_{min} be the minimum integer such that the total number of leaked bits in the top γ_{min} EKO's in S is ≥ 160
- 12 $i = \gamma_{min}$
- 13 $maxIter = |S|$
- 14 **while** $i \leq maxIter$ AND $keyNotFound = true$ **do**
- 15 Create an SVP instance using the top i EKO's from S
- 16 Solve the SVP instance, compute the DSA private key and verify its correctness
- 17 **if** *computed key is correct* **then**
- 18 $keyNotFound = false$
- 19 **else**
- 20 $i \leftarrow i + 1$
- 21 **end**
- 22 **return** *computed key*

We refer to each input of an SVP instance as an *Ephemeral Key Occurrence (EKO)*. Each EKO comprises the positions and values of a “large” interior block and the rightmost one. By large is meant a block size of 5 or more. An ephemeral key may contribute more than a single EKO to the SVP instance in which case the rightmost block will be repeated. In the toy example of Sect. 3.1, the ephemeral key contributes two EKOs since there are two “large” interior blocks, one of size 5 and the another of size 6. Each EKO includes the rightmost block (length = 2). The total number of bits leaked by these two EKO’s is $5 + 6 + 2 = 13$. On average, an ephemeral key contributes 3–4 EKOs.

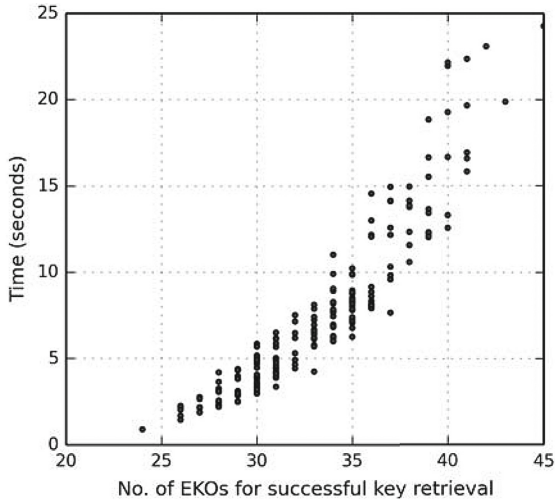


Fig. 3. Scatter plot of No. of EKOs required for success and corresponding time ($n = 140$) (Algorithm 1)

From each such triplet $\langle message, signature, SMsequence \rangle$, we identify all known “large” blocks and construct a set of EKOs. Blocks that reside in the leftmost 25% of the ephemeral key are excluded from consideration since the error probabilities of those blocks average 40% (as discussed in Sect. 3.2). The remainder of EKOs are sorted in descending order of the sum of the lengths of the interior block and rightmost block (the sorted list is denoted S).

To obtain the DSA signing key, we draw the top γ_{min} EKOs from S where γ_{min} is the smallest integer such that the γ_{min} EKOs collectively leak a total of 160 or more bits of their ephemeral keys (Algorithm 1). We build an SVP instance from the input provided by these EKOs, solve the SVP problem, compute the DSA key and verify its correctness. If we fail, we continue this process but we now include the next EKO from S while retaining the existing EKOs. This is repeated until we obtain the correct DSA key. This simple procedure yielded the DSA key 89% of the time with 30 signatures and 100% of the time with 40 signatures with the number of iterations capped at 100. Figure 3 is a scatter plot showing the number of EKOs required to obtain the DSA key (using Algorithm 1) and the corresponding execution time.

Algorithm 2. Key Retrieval using noisy side channel - Approach 1

Input: τ triplets of \langle message, signature, EKO \rangle , one per EKO in T
Output: DSA Private Key

```

/*  $T$  comprises the top  $\tau$  EKO's in set  $S$  (from Algorithm 1). */
/*  $A$  is the set of EKO's actually used in the instance of an SVP,
    $A \subseteq T$ . */
1 keyNotFound = true
2 iterMax = 10000 //maximum possible attempts
3 iter = 1
4 do
5     Populate  $A$  with the fewest number of randomly selected EKO's from  $T$  such
        that the total number of bits leaked from the EKO's in  $A > 160$ 
6      $T' = T - A$ 
7     do
8         Build an SVP instance using the EKO's from  $A$ 
9         Solve SVP and compute the DSA key
10        if computed key = correct key then
11            | keyNotFound = false
12            | return computed DSA key and  $A$ 
13        else
14            | Select a random EKO,  $e$ , from  $T'$ 
15            |  $A = A \cup \{e\}$ 
16            |  $T' = T' - \{e\}$ 
17        while keyNotFound = true AND  $|A| \leq \gamma_{max}$ ;
18        iter  $\leftarrow$  iter + 1
19 while keyNotFound = true AND iter < iterMax;

```

4.4 Retrieving the DSA Key from a Noisy Side Channel

Given SM sequences from a noisy side channel, we follow two approaches. In both cases, we populate a set, T with the top τ EKO's in S (created in Algorithm 1). The first approach (Algorithm 2) is a direct extension of Algorithm 1.

A single iteration of the outer loop of Algorithm 2 proceeds as follows. First, a set, A is populated by randomly selecting γ_{min} EKO's from T (as before, γ_{min} denotes the smallest integer such that the γ_{min} EKO's collectively leak a total of 160 or more bits of their ephemeral keys). An SVP instance is created and solved and the DSA key is computed. If the correct key is not obtained, another EKO from the set T is randomly selected and added to the set A (while retaining the existing ones selected earlier). As before, the DSA key is computed after solving an SVP instance. This procedure is repeated, each time incrementing the cardinality of A by 1, until the DSA key is computed or a certain stipulated maximum number of iterations, γ_{max} of the inner loop are executed. γ_{max} is experimentally determined from Algorithm 1 and is sufficiently large to virtually guarantee success with error-free EKO's (Table 1).

A major difference between Algorithms 1 and 2 is the way the EKO's are chosen. In the former, the EKO's are chosen sequentially from the list S . In the latter, they are randomly selected from the set, T . The outer loop is repeated, each time starting with a random subset of γ_{min} EKO's drawn from T . We refer to each iteration of the outer loop as an attempt. In Sect. 5.1, we derive a distribution for the number of attempts, ϑ , required for successful key retrieval.

We next outline an alternative approach to obtain the DSA key given the SM sequences harvested from a noisy channel (Algorithm 3). In each iteration of the outer loop, we randomly choose γ EKO's from the set T to be used as input to an SVP instance. We solve the SVP and compute the DSA key. If the correct key is not recovered, we work with the same set of EKO's but now these are randomly shuffled and then input to an SVP instance. This process of randomly shuffling the EKO's and solving an SVP instance with the shuffled set is repeated ρ times (or fewer if the DSA key is found earlier). The value of ρ required to guarantee a sufficiently high probability of success is addressed in the next section.

Algorithm 3. Key Retrieval using a Noisy Side Channel - Approach 2

Input: Triplets <message, signature, SM sequence> one per EKO in T

Output: DSA Private Key

```

1 keyNotFound = true
2 iterMax = 10000 //maximum possible attempts
3 iter = 1
4 do
5   | randomly select  $\gamma$  EKO's from  $T$ 
6   | inner=0
7   | do
8   |   | Create an SVP instance using the  $\gamma$  EKO's
9   |   | Solve SVP and compute DSA key and verify correctness
10  |   | if computed key = correct key then
11  |   |   | keyNotFound = false
12  |   |   | return computed DSA key
13  |   | else
14  |   |   | randomly shuffle the  $\gamma$  EKO's
15  |   |   | inner  $\leftarrow$  inner + 1
16  |   | while keyNotFound = true AND inner <  $\rho$ ;
17  |   | iter  $\leftarrow$  iter + 1
18 while keyNotFound = true AND iter < iterMax;

```

If even after ρ shufflings, we do not recover the DSA key, we suspect that one or more of the participating EKO's is in error. We therefore proceed to the next iteration of the outer loop but with a fresh set of randomly selected EKO's from T . The number of attempts (iterations of the outer loop), on average, $\bar{\vartheta}$ required for success is investigated next.

5 Modeling, Results and Analysis

The success probabilities of an attempt in Algorithms 2 and 3 are first derived. This is followed by a comparison of the number of attempts and total execution time as a function of number of signatures with Algorithms 2 and 3. Experimentally obtained values are also used to compare the various options.

5.1 Modeling Success Probability

Success Probability of Algorithm 2: Algorithm 2 involves creating and solving a sequence of SVP instances each with inputs from $\gamma_{min}, \gamma_{min} + 1, \dots, \gamma_{max}$ EKO's.

Let $p_{i,n}$ be the experimentally determined probability of success in an iteration of the inner loop of Algorithm 2 involving i error-free EKO's (and assuming that all previous inner loop iterations of this attempt have failed). With error-free EKO's, $\sum_{i=\gamma_{min}}^{\gamma_{max}} p_{i,n} \approx 1$. However, with SM sequences derived from a noisy channel, the success of an iteration is conditioned on the probability, p_i , that all the i participating EKO's are error-free. All i EKO's are drawn from the set, T of τ EKO's. Given that e is the probability that an EKO is in error, the number of error-free EKO's in T is, on average, $\tau(1 - e)$. There are $\binom{\tau(1-e)}{i}$ ways of choosing i EKO's from the set of $\tau(1 - e)$ error-free EKO's and $\binom{\tau}{i}$ ways of choosing i EKO's from a set of τ EKO's. So,

$$p_i(\gamma) = \frac{\binom{\tau(1-e)}{i}}{\binom{\tau}{i}}$$

Summing over the success probabilities of each inner loop iteration, we obtain the success probability of an attempt, P_s

$$\begin{aligned} P_s &= \sum_{i=\gamma_{min}}^{\gamma_{max}} p_{i,n} \times \frac{\binom{\tau(1-e)}{i}}{\binom{\tau}{i}} \\ &= \sum_{i=\gamma_{min}}^{\gamma_{max}} p_{i,n} \times \frac{\prod_{j=0}^{i-1} (\tau(1-e) - j)}{\prod_{j=0}^{i-1} (\tau - j)} \end{aligned}$$

Success Probability of Algorithm 3: Unlike in Algorithm 2, each SVP instance in Algorithm 3 is constructed using the same number of EKO's, γ . The randomly selected set of EKO's in each attempt is subjected to the same number of random shufflings, ρ . The values of γ and ρ are experimentally determined and represent a tradeoff between higher success probability and lower execution

time. In particular, for the values of γ and ρ selected in Table 1, the probability of success assuming error-free EKO's is 0.9.

With the same logic used to analyze Algorithm 2, the success probability of an attempt (iteration of the outer loop) is

$$\begin{aligned}
 P_s &= 0.9 \times \frac{\binom{\tau(1-e)}{\gamma}}{\binom{\tau}{\gamma}} \\
 &= 0.9 \times \frac{\prod_{j=0}^{\gamma-1} (\tau(1-e) - j)}{\prod_{j=0}^{\gamma-1} (\tau - j)}
 \end{aligned}$$

5.2 Results and Analysis

The notation and values of various parameters used in the algorithms, experiments and model are shown in Table 1. Figure 4 shows the distribution of the Number of attempts to achieve success, (ϑ) in each of four cases:

- (i) Algorithm 2, # signatures (n) = 140
- (ii) Algorithm 2, n = 100
- (iii) Algorithm 3, n = 140
- (iv) Algorithm 3, n = 100

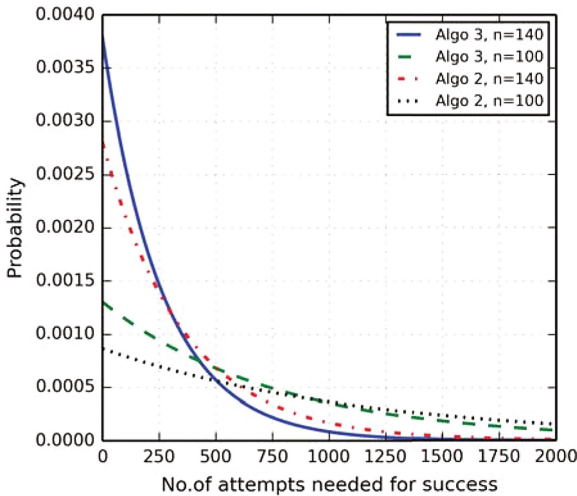


Fig. 4. Distribution of No. of attempts for success (ϑ)

It is clear that ϑ with 100 signatures is more broadly distributed while ϑ with 140 signatures is much more narrowly focused at smaller values. With 140 signatures, there is a larger pool of high-quality EKO's (those with larger known blocks). Such EKO's leak out a larger number of bits in their ephemeral keys and so fewer EKO's are required to create SVP instances. The probability that all EKO's in a set are error-free is inversely related to the cardinality of the set leading to a higher probability of a successful attempt with 140 signatures.

The expected value of ϑ is

$$\begin{aligned}\bar{\vartheta} &= \sum_{i=1}^{\infty} iP_s (1 - P_s)^{i-1} \\ &= \frac{1}{P_s}\end{aligned}$$

Given the experimentally observed average error rate, $e = 0.128$, ϑ is higher with Algorithm 2 compared to that with Algorithm 3 (Table 2). This is because the average number of EKO's used to build an SVP instance with Algorithm 2 is higher leading to higher probability of error in the ensemble of EKO's and hence failed attempts.

Table 2. Performance as a function of No. of signatures and algorithm

Metric	No. of signatures (n) = 140		No. of Signatures (n) = 100	
	Algorithm 2	Algorithm 3	Algorithm 2	Algorithm 3
Average No. of attempts to succeed (Model)	356	264	1157	767
Average No. of attempts to succeed (Expt.)	418	412	1636	645
Average Time per attempt in sec. (Expt.)	47.1	37.2	84.5	114.7
Total Time in hours on a single core (Model)	4.7	2.7	21.2	24.4

We ran Algorithms 2 and 3 on 10–15 samples of 100 and 140 signatures each and recorded the number of attempts required to obtain the DSA key. The average values are shown in Table 2. The discrepancy between theoretical and experimental values is not insignificant in two of the four cases. We attribute this, in part, to the limited set of experiments performed. More important, is the considerable dependence of the model results on experimentally observed/determined parameters such as e and ρ . For example, while the model uses a single average error probability, the actual value of e varies from sample to sample. Nevertheless, the main conclusions to be drawn from, both, model and experiment results are similar – $\bar{\vartheta}$ increases greatly with a more limited pool of available signatures and Algorithm 3 outperforms Algorithm 2.

In addition to $\bar{\vartheta}$, the average time per attempt also determines the execution time for DSA key retrieval (Table 2). These times were measured on an Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz with 4 GB memory running Linux kernel 3.16.35. The maximum time per (unsuccessful) attempt is 114.7 s taken by Algorithm 3 with 100 signatures. In this case, the number of random shufflings performed is 100 (Table 2) compared to only 50 with 140 signatures. Also, the number of EKO's used in each attempt is 34 compared to 30 with 140 signatures.

With Algorithm 2 and 100 signatures, the time per attempt is 84.5 s versus 47.1 s with 140 signatures. Each unsuccessful attempt in the former involves solving 60 SVP instances while the corresponding number in the latter is 50 (Table 2). Moreover, the 10 extra SVP solutions involve lattices with higher dimensions and SVP solution time increases superlinearly with increasing number of dimensions (Fig. 3).

Table 2 also shows the average time to retrieve the DSA key in each case based on the model estimates of $\bar{\vartheta}$ and the experimentally measured average time per attempt. Both factors that dictate total execution time increase greatly as the number of signatures decrease. Hence it is not surprising that total execution time on a single core with 140 signatures is 2–5 h but it is over 20 h with 100 signatures. Our experiments were performed on cores of machines with diverse speeds, so comparing such values would be misleading. Nevertheless, the execution times measured by us corroborate the model estimates of >20 h with 100 signatures and considerably lower times with 140 signatures.

Finally, Fig. 5 shows the extreme sensitivity of $\bar{\vartheta}$ to error probability. The number of attempts required to achieve success increases exponentially with the error probability. For example, an increase of e from 0.1 to 0.15 results in a “10-fold” increase in $\bar{\vartheta}$ with 140 signatures. The increase in $\bar{\vartheta}$ is even more dramatic with 100 signatures - the number of attempts to achieve success increases from

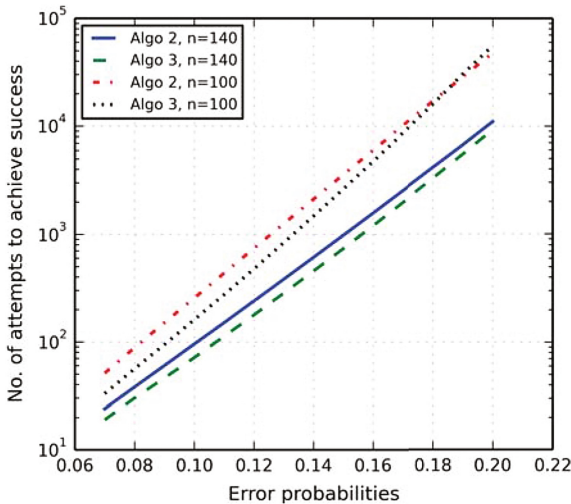


Fig. 5. Number of attempts for success v/s Error probabilities

255 to 3571. While Algorithm 3 is conspicuously better than Algorithm 2 at lower error rates, the difference seems to narrow (and even reverse for $n = 140$) as e approaches about 0.2.

6 Related Work

Several attacks have been crafted, in the past on DSA and ECDSA using lattices when partial information on the per signature random ephemeral key is available. Howgrave-Smart [5] explained the private key recovery with key size of 160 bits. According to them, when 8 LSB bits of the ephemeral keys are known, with 30 signatures, the private key can be retrieved using the CVP approach. They also explain the lattice attack using CVP approach when the known information is spread across the ephemeral key. In the latter case, [5] when the known information is available in two blocks, 12 signatures are required with 16 known bits across these blocks. In a further work Nguyen-Spharliniski [6, 13] formulated the attack using Hidden Number Problem(HNP) as in [18] which again is solved using CVP solvers such as Babai's Nearest Plane Algorithm [14]. According to [6], it requires 5 known LSBs for a 100% successful key retrieval and yielded 90% success when only 4 LSBs are known. In [19], Liu-Nguyen implemented a method called GNR Pruning [20] to retrieve the private key when only 2 LSBs are known with 23% success probability. [7] used HNP to retrieve the ECDSA private key when variable LSB bits are used per ephemeral key. Similarly, [8] addressed the problem when the known bits are in the middle blocks. [12] extends the attack on DSA on real time OpenSSL data by considering only the rightmost(LSB) blocks of the ephemeral key. They require 280 signatures intercepted from SSH in successful private key retrieval.

We used Flush + Reload based side channel attack which was first introduced in [10] for attacking T-table implementation of AES on single core. Later it was named by [11] to conduct multi-core attack by targeting last level cache (L3) to extract private encryption key from RSA implementation in GnuPG. They also demonstrated Flush + Reload attack in Cross-VM scenario with 96.7% success. The same approach was used by Irazoqui et al. [21] to take advantage of deduplication mechanism called Transparent Page Sharing and demonstrated a full key recovery from AES T-table implementation. Lipp et al. [22] performed Flush + Reload on ARMv8-A and ARMv7-A processors on Android devices and recovered the AES key.

7 Summary and Concluding Remarks

The SM sequences for multiple signature operations are captured by the spy through a cache-based side channel attack. From these, we derive discontinuous known blocks of each ephemeral key. These are then input to an SVP instance from which the DSA private key is computed. Through experiments, the error probability in identifying the position and length of such blocks was estimated. A strong positive correlation between the error probability and distance from

the LSB end was observed. Based on this observation, it was decided to exclude blocks within 40 bits of the MSB end as input to an SVP instance.

We sorted all EKO's in descending order of the sum of the lengths of the interior block and the rightmost block and set aside the top τ for further consideration. We randomly selected a minimum number of EKO's from these, incrementally adding an EKO until solution of an SVP instance retrieved the DSA key or we reached a stipulated maximum number of EKO's. To handle noise, we implemented multiple rounds of the above until we retrieved the DSA private key. We developed a model to predict the number of rounds required as a function of number of signatures and error rate. The model suggests that the number of rounds increases exponentially with the error rate. In practice, we were able to recover the DSA key on a single core with 140 signatures in about 5 h on average.

Our focus in this paper was to retrieve the DSA private key with inputs received from a noisy side channel with performance being a secondary goal. Accordingly, the algorithms employed here are a trade-off between simplicity and efficiency. The total number of bits leaked by the EKO's in a successful run of our experiments ranges between 250 and 450. This suggest considerable room for improvement through techniques in [19,23]. We also feel that it is possible to decrease the error rate of the inputs received from the cache-based side channel by more refined measurements. Given that the execution time for key retrieval is critically dependent on error rate, efforts in this direction may yield big dividends.

Last, as with most side channel attacks, simple countermeasures may prove highly effective. For example, the edge of the sliding window could be aligned on static 4-bit boundaries. Thus, the SM sequence would be fixed and reveal nothing about the ephemeral key. The performance advantage of using windowing would be retained though it would be marginally less than that with the SWE implementation currently used. It would also be interesting to study the effect of larger window sizes (say 5 or 6) on the success of these attacks.

References

1. FIPS: 186–2. Digital Signature Standard (DSS). National Institute of Standards and Technology (NIST), vol. 20, p. 13 (2000)
2. Ashokkumar, C., Giri, R.P., Menezes, B.: Highly efficient algorithms for AES key retrieval in cache access attacks. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 261–275. IEEE (2016)
3. Durfee, G., Nguyen, P.Q.: Cryptanalysis of the RSA schemes with short secret exponent from Asiacypt 1999. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 14–29. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_2
4. Boneh, D., Durfee, G., Frankel, Y.: An attack on RSA given a small fraction of the private key bits. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 25–34. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49649-1_3
5. Howgrave-Graham, N.A., Smart, N.P.: Lattice attacks on digital signature schemes. *Des. Codes Crypt.* **23**(3), 283–290 (2001)

6. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the digital signature algorithm with partially known nonces. *J. Cryptol.* **15**(3), 151–176 (2002)
7. Bengier, N., van de Pol, J., Smart, N.P., Yarom, Y.: “Ooh Aah... Just a Little Bit”: A small amount of side channel can go a long way. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 75–92. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_5
8. van de Pol, J., Smart, N.P., Yarom, Y.: Just a little bit more. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 3–21. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16715-2_1
9. Fan, S., Wang, W., Cheng, Q.: Attacking openssl implementation of ECDSA with a few signatures. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1505–1515. ACM (2016)
10. Gullasch, D., Bangerter, E., Krenn, S.: Cache games-bringing access-based cache attacks on AES to practice. In: 2011 IEEE Symposium on Security and Privacy (SP), pp. 490–505. IEEE (2011)
11. Yarom, Y., Falkner, K.: Flush+reload: A high resolution, low noise, L3 cache side-channel attack. In: USENIX Security Symposium, pp. 719–732 (2014)
12. Pereida García, C., Brumley, B.B., Yarom, Y.: Make sure DSA signing exponentiations really are constant-time. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1639–1650. ACM (2016)
13. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Des. Codes Crypt.* **30**(2), 201–217 (2003)
14. Babai, L.: On lovaszlattice reduction and the nearest lattice point problem. *Combinatorica* **6**(1), 1–13 (1986)
15. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. *Math. Ann.* **261**(4), 515–534 (1982)
16. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.* **66**(1–3), 181–199 (1994)
17. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_1
18. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes. In: Kobitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 129–142. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_11
19. Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: an update. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 293–309. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36095-4_19
20. Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 257–278. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_13
21. Irazoqui, G., Inci, M.S., Eisenbarth, T., Sunar, B.: Wait a minute! a fast, cross-VM attack on AES. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 299–319. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11379-1_15
22. Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., Mangard, S.: Armageddon: Cache attacks on mobile devices. In: USENIX Security Symposium, pp. 549–564 (2016)
23. Lindner, R., Peikert, C.: Better key sizes (and Attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19074-2_21

Efficient Construction of Diamond Structures

Ariel Weizmann¹ , Orr Dunkelman² , and Simi Haber¹

¹ Department of Mathematics, Bar-Ilan University, Ramat-Gan, Israel

² Computer Science Department, University of Haifa, Haifa, Israel
orrd@cs.haifa.ac.il

Abstract. A cryptographic hash function is a function $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$, that takes an arbitrary long input and transforms it to an n -bit output, while keeping some basic properties that ensure its security. Because they are very useful in computer security, cryptographic hash functions are amongst the most important primitives in the modern cryptography.

The Merkle-Damgård structure is an iterative construction for transforming a compression function $f : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$ into a hash function, and it is widely used by different hash functions such as MD4, MD5, SHA0 and SHA1. Some generic attacks on this structure were presented in the last 15 years. Some of these attacks use the *diamond structure*, first introduced by Kelsey and Kohno in the herding attack. This structure is a complete binary tree that allows 2^k different inputs to lead to the same hash value, and it used in numerous attacks on the Merkle-Damgård structure. Following the herding attack, other papers analyzed and optimized the *diamond structure*. The best time complexity of constructing a *diamond structure* to date is about $a \cdot 2^{\frac{n+k}{2}+2}$ for $a \approx 2.732$.

In this work we suggest a new and simple method for constructing a diamond structure with better time complexity of $c \cdot 2^{\frac{n+k}{2}+2}$ for $c \approx 1.254$. We present a pseudo-code for this new method, and a recursive formulation of it. We also present analysis supported by experiments of our new method.

1 Introduction

Cryptographic hash functions are one of the important basic primitives in cryptography. Their importance is reflected in their wide use: digital signatures, hashed passwords, message authentication code (MAC), etc.

Design and cryptanalysis of hash functions has become one of the hottest research topics in the last fifteen years, when a series of groundbreaking works showed that some of the hash function designs (including Merkle-Damgård construction) are theoretically insecure [1, 10, 15, 17, 18], and that most of used hash functions (including SHA0, SHA1, MD4, MD5, RIPEMD, etc.) are theoretically (and some of them also practically) insecure [5, 19, 24–29]. These results called for rethinking of the hash functions design methodologies, and invited new designs and their analysis. As part of this rethinking, the National Institute of Standards

and Technology (NIST) announced a selection process of a new hash function standard called SHA3, which culminated in the choice of Keccak [4] as SHA3 function in October 2012. The SHA3 process gave rise to numerous new design methodologies and continuously developing cryptanalytic techniques.

One of the main sources for comparison between design strategies are generic attacks. While usually non-practical, they point out structural weaknesses in a strategy that may make us prefer a more conservative (or just a different) design. One of the basic designs of numerous hash functions is the Merkle-Damgård structure [9, 22], which, given a compression function $f : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$, creates a cryptographic hash function $\mathcal{MDH}^f : \{0, 1\}^* \rightarrow \{0, 1\}^n$, so that the hash function has certain security properties. Numerous generic attacks were presented against this construction, e.g., Joux's multicollision attack on iterative hash functions [15], the expandable messages attack of Kelsey and Schneier [18] and the herding attack of Kelsey and Kohno [17]. Naturally, generic attacks are used at complex algorithms and designs, often become used by other attacks.

This work improves the complexity of attacks based on the *diamond structure*, first introduced by Kelsey and Kohno in the herding attack [17]. Kelsey and Kohno calculated the diamond construction complexity by intuitive reasoning concluded that building a diamond structure of 2^k leaves takes $2^{\frac{n+k}{2}+2}$ compression function calls. Blackburn et al. [6] showed that their calculation is wrong, and the real complexity, by the method presented by Kelsey-Kohno, is actually $\sqrt{k} \cdot 2^{\frac{n+k}{2}+2}$ compression function calls. In [21] Kortelainen and Kortelainen suggested a new method to construct the diamond in $a \cdot 2^{\frac{n+k}{2}+2}$ compression function calls, for $a = 2.732$. In this paper we suggest a new method for constructing the diamond in $c \cdot 2^{\frac{n+k}{2}+2}$ compression function calls, for $1 \leq c \leq 2$. Our experiments show that for our algorithm $c = 1.254$, suggesting that the original claims of Kelsey and Schneier were of sufficient precision. The advantage of our work over the previous is not only the time complexity improvement, but also the algorithmic improvement: While the Kortelainen algorithm is very complex, our algorithm is simple and intuitive.

This paper is organized as follows: Sect. 2 gives notations and definitions used in this paper. In Sect. 3 we quickly recall the herding attack, and most importantly, the construction of diamond structures. We discuss the different methods to construct a diamond structure in Sect. 4. Our new ideas on how to efficiently build a diamond structure are given in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Notations and Definitions

Definition 1. *A cryptographic hash function is a function $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$, that takes an arbitrary length input and transforms it to an n -bit output such that $H(x)$ can be computed efficiently, while the function has three basic security properties:*

1. Collisions resistance: It is hard to find (with high probability) an adversary that could find two different messages M, M' such that $H(M) = H(M')$ in less than $\mathcal{O}(2^{n/2})$ calls to $H(\cdot)$.
2. Second pre-image resistance: Given h, M such that $H(M) = h$, an adversary cannot find (with high probability) an additional message $M' \neq M$ such that $H(M') = h$ in less than $\mathcal{O}(2^n)$ calls to $H(\cdot)$.
3. Pre-image resistance: Given a hash value h , an adversary cannot find (with high probability) any message M such that $H(M) = h$ in less than $\mathcal{O}(2^n)$ calls to $H(\cdot)$.

Definition 2 (Merkle-Damgård structure (\mathcal{MDH})). *The Merkle-Damgård structure [9, 22] is a structure of an iterative hash function, based on a compression function $f : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$. The compression function takes an n -bit chaining value and an m -bit message block and transforms them into a new n -bit chaining value, keeping the three basic properties described above. In order to hash a whole message M , the following steps are required (let b be the number of bits in the message, and ℓ be the number of bits used to encode the message length in bits¹):*

1. Padding step:
 - (a) Concatenate ‘1’ to the end of the message.
 - (b) Pad a sequence of $0 \leq k < m$ zeros, such that $b + 1 + k + \ell \equiv 0 \pmod{m}$.
 - (c) Append the message with the original message length in bits, encoded in ℓ bits.
2. Divide the message to blocks of m bits, so if the length of padded message is $L \cdot m$ then

$$M = M_0 || M_1 || \dots || M_{L-1}.$$
3. The iterative chaining value h_i starts with a constant IV , defined as h_{-1} of the hash function, and it updated in every iteration, according to the appropriate message block M_i , to new chaining value: $h_i = f(h_{i-1}, M_i)$.
4. The output of this process is: $\mathcal{MDH}^f(M) = h_{L-1}$.

The structure of the Merkle-Damgård hash function is depicted in Fig. 1. Merkle [22] and Damgård [9] proved that if the compression function is collision-resistant then the whole structure (when the padded message includes the original message length) is also collision-resistant.

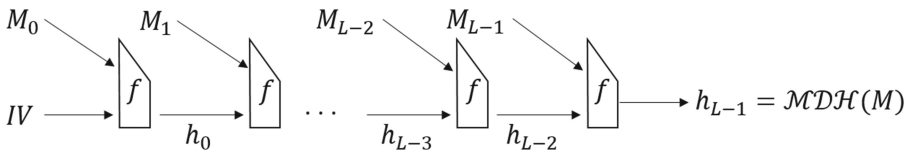


Fig. 1. The Merkle-Damgård structure

¹ It is common to set $2^\ell - 1$ as the maximal length of a message.

3 The Herding Attack and the Diamond Structure

A well known generic attack on the Merkle-Damgård structure is the Kelsey-Kohno herding attack [17]. This attack has two phases. In the first phase, the adversary performs some precomputation and commits to a hash value h . In the second phase, given a prefix P , he finds a suffix S s.t. $H(P||S) = h$.

In the precomputation the adversary constructs a complete binary tree called a *diamond structure*. This structure allows 2^k sequences of k message blocks to iteratively lead to the same chaining value. This may seem related to Joux’s multicollision attack [15] where 2^k different message blocks result in the same chaining value. However, in the case of Joux’s multicollision attack, all these 2^k options start from the same chaining value, whereas in the diamond structure, there are 2^k different starting chaining values.

To construct the *diamond structure*, the adversary starts with 2^k different chaining values, and looks for collisions between pairs of these chaining values to map them down to 2^{k-1} chaining values. In Sect. 4 we discuss different methods to do so. He repeats this process k times, s.t. in every iteration $1 \leq i \leq k$ he maps the 2^{k-i+1} chaining values he received at the end of the previous iteration down to 2^{k-i} chaining values. The output of this process, after k iterations, is a single hash value h . Figure 2 illustrates this structure for $k = 3$, when the arrows represent message blocks, and the values $h_{i,j}$ represent chaining values. This structure generates a multicollision of 2^k messages, when k is the diamond width. After the construction of the diamond structure the adversary commits the output h .² According to Kelsey and Kohno, the work done to construct the diamond structure is about $2^{\frac{n+k}{2}+2}$ compression function calls, and we will discuss it later in Sect. 4.

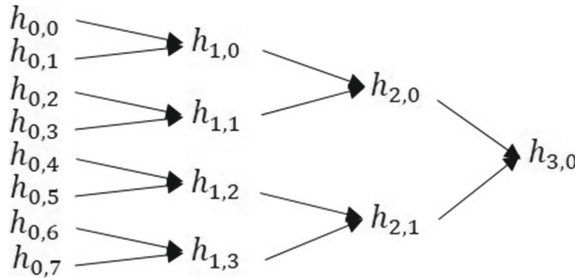


Fig. 2. Diamond structure for $k = 3$

In the second phase, the adversary is challenged with a prefix P , and he has to find a suffix S yielding the desired result $H(P||S) = h$. To do so, he looks

² He should consider the length of the prefix P , or at least the maximum length of it, and if the real length is less than he considered, he can add some blocks to the M_{link} block, which will be defined in the second phase.

for a message block M_{link} that links the chaining value yielded from the prefix P to one of the leaves $\{h_{0,i}\}$ of the diamond structure. Given a chaining value h_{0,i_0} , traversing the tree from this leave to the root, appending the message blocks from each edge, leads the string Q , which creates, together with M_{link} , the desired suffix $S = M_{link}||Q$. Since the diamond has 2^k leaves, the work done to find the M_{link} block is about 2^{n-k} compression function calls, and thus the total work (according to Kelsey and Kohno) is about

$$2^{\frac{n+k}{2}+2} + 2^{n-k}.$$

We note that the diamond structure was later used in [1,3] to offer second pre-image attacks against Merkle-Damgård hash functions, including dithered hash functions [23].

4 Previous Methods for Constructing Diamond Structure

After Kelsey and Kohno published their attack based on the diamond structure and suggested their method and its analysis [17], several papers were published on methods for constructing and analyzing the diamond structure. We now discuss them.

4.1 Kelsey-Kohno's Method

In [17] Kelsey and Kohno suggest a method for constructing a diamond structure. We now describe their method for the first level of the diamond, and the application for the other levels is immediate.

Given 2^k starting chaining values, $\{h_{0,0}, h_{0,1}, \dots, h_{0,2^k-1}\}$, the adversary should find 2^{k-1} collisions between pairs of them to map them down to 2^{k-1} new chaining values. i.e., he should find a partition to pairs of $\{h_{0,0}, h_{0,1}, \dots, h_{0,2^k-1}\}$, and for each pair $(h_{0,i}, h_{0,j})_{i \neq j}$ he should find message blocks M, M' (not necessarily different) such that $f(h_{0,i}, M) = f(h_{0,j}, M')$.

If he fixes pairs of them, like $\{(h_{0,0}, h_{0,1}), (h_{0,2}, h_{0,3}), \dots, (h_{0,2^k-2}, h_{0,2^k-1})\}$, then he should generate about $2^{\frac{n}{2}}$ candidate message blocks for each pair to find a collision, and thus the work done to find 2^{k-1} collisions is about $2^{k-1} \cdot 2^{\frac{n}{2}} = 2^{\frac{n}{2}+k-1}$ compression function calls.

Instead, he generates about $2^{\frac{n-k+1}{2}}$ candidate message blocks from each starting chaining value $h_{0,i}$, and then looks for collisions between all the possibility pairs dynamically. When he finds a collision, i.e., two starting chaining values $h_{0,i}, h_{0,j}$ and two message blocks M, M' s.t. $f(h_{0,i}, M) = f(h_{0,j}, M') = h$, he chooses these message blocks for these chaining values, and takes the new chaining value h for the next level of the diamond.

Kelsey and Kohno expected to find 2^{k-1} such collisions (we present their computation in Sect. 4.2), i.e., $2^k \cdot 2^{\frac{n-k+1}{2}} = 2^{\frac{n+k+1}{2}}$ message blocks should be sufficient to map all the 2^k starting chaining values into 2^{k-1} new chaining values will be in the next level of the diamond.

4.2 Kelsey-Kohno’s Original Analysis

Kelsey and Kohno calculate the complexity of the diamond construction, and conclude the following:

“The work done to build the diamond structure is based on how many messages must be tried from each of 2^k starting values, before each has collided with at least one other value. Intuitively, we can make the following argument, which matches experimental data for small parameters: When we try $2^{\frac{n}{2} + \frac{k}{2} + \frac{1}{2}}$ messages spread out from 2^k starting hash values (lines), we get $2^{\frac{n}{2} + \frac{k}{2} + \frac{1}{2} - k}$ messages per line, and thus between any pair of these starting hash values, we expect about $(2^{\frac{n}{2} + \frac{k}{2} + \frac{1}{2} - k})^2 \times 2^{-n} = 2^{n+k+1-2k-n} = 2^{-k+1}$ collisions. We thus expect about $2^{-k+k+1} = 2$ other hash values to collide with any given starting hash value” [17].

According to [17], if the adversary generates $2^{\frac{n-k+1}{2}}$ message blocks for each starting chaining value then the probability of a collision between any two starting chaining values is 2^{-k+1} . Thus, given a starting chaining value, since it could collide with any other starting chaining value, he expects about $2^{-k+1+k} = 2$ collisions. Since for any starting chaining value he expects at least one collision, he expects to map all the 2^k starting chaining values down to 2^{k-1} new chaining values.

Now, to construct the whole diamond he should repeat this work k times, for every $1 \leq i \leq k$, and thus the total complexity is about

$$\sum_{i=1}^k 2^{\frac{n+i+1}{2}} \approx 2^{\frac{n+k}{2} + 2}.$$

4.3 On the Inaccuracy of Kelsey-Kohno’s Analysis

Blackburn et al. [6] show that although this calculation is correct, their conclusion is wrong. They suggest to model the Kelsey-Kohno’s method by the Erdős-Rényi random graph $G(n, p)$. The $G(n, p)$ is a random graph with n vertices, and there is an edge between any two vertices with probability p independently of other edges. In Kelsey-Kohno’s case we get $G(2^k, 2^{-k+1})$, where $V = \{h_{0,0}, h_{0,1}, \dots, h_{0,2^k-1}\}$ and $(h_{0,i}, h_{0,j}) \in E$ if and only if there exist two message blocks M, M' s.t. $f(h_{0,i}, M) = f(h_{0,j}, M')$, and it happens with probability of 2^{-k+1} , as described in Kelsey-Kohno’s calculation. In this perspective, mapping the 2^k starting chaining values into 2^{k-1} new chaining values is equal to the existence of a perfect matching in G . In [11–13] Erdős and Rényi show that for a random graph $G(n, p)$, if $p > \frac{(1+\epsilon)\ln n}{n}$ then G will almost surely be connected and will contain a perfect matching, and if $p < \frac{(1-\epsilon)\ln n}{n}$ then G will almost surely contain isolated vertices and thus it will not contain any perfect matching. It means that $p = \frac{\ln n}{n}$ is a sharp threshold for the existence of a perfect matching in $G(n, p)$. In Kelsey-Kohno’s case we get that (for $k > 2$)

$$p = 2^{-k+1} < k \cdot \ln 2 \cdot 2^{-k} = \frac{\ln(2^k)}{2^k}.$$

Thus, the graph will almost surely contain isolated vertices and thus it will not contain any perfect matching.

Hence, despite of the correct conclusion of Kelsey and Kohno that about $2^{-k+k+1} = 2$ edges for each vertex are expected, there will be many isolated vertices.³

To fix the problem, Blackburn et al. [6] conclude that we should generate about $\sqrt{k} \cdot 2^{\frac{n-k}{2}}$ message blocks from each vertex. Thus, to construct the entire diamond structure about

$$\sum_{i=1}^k \sqrt{i} \cdot 2^{\frac{n+i}{2}} \approx \sqrt{k} \cdot 2^{\frac{n+k}{2}+2}$$

message blocks are needed.⁴

4.4 Kortelainen-Kortelainen’s Method

In [21] Kortelainen and Kortelainen suggest a new method to construct the diamond structure. Their general idea is to divide the construction into steps such that in every step exactly two vertices are matched to a single chaining value. If a vertex is matched they stop generating more message blocks for it. We describe here their algorithm for the first level of the diamond, the application for the other levels is immediate.

They divide the process into k phases (in reduced order), such that in every phase $2 \leq j \leq k$ they match exactly 2^{j-1} vertices, and in the last phase $j = 1$ they match the two remaining vertices, so at the end of these phases all the vertices are matched. Every phase $2 \leq j \leq k$ is divided into 2^{j-2} steps, such that in every step they match exactly two vertices to a new chaining value. The last step is done by finding a collision between the last two remaining vertices. Before the process, an initialization phase is performed as follow: Given the initial hash values (denoted by $A_{k,0}$), create a set $M_{k,0}$ of $2^{\frac{n-k}{2}-1}$ message blocks, such that the cardinality of $H_{k,0} := f(A_{k,0}, M_{k,0}) = \{f(a, m) | a \in A_{k,0}, m \in M_{k,0}\}$ is $2^{\frac{n+k}{2}-1}$, i.e., there are no collisions by these message blocks.⁵ In Addition, they

³ The degree of each vertex follows a Poisson distribution with a mean of 2. Thus, for each vertex, the probability that it is an isolated vertex is e^{-2} , and thus we expect to about $2^k \cdot e^{-2}$ isolated vertices.

⁴ Blackburn et al. [6] discuss another model to represent the diamond construction: *Sampling With Replacement Random Intersection Graph* $G_{SWR}(\nu, m, L)$ random graph, defined as follow: Let V be a set of vertices where $|V| = \nu$ (in our case $\nu = 2^k$), and F be a colors set where $|F| = m$ (in our case $m = 2^n$). For each vertex $v \in V$ generate a subset $F_v \subset F$ by sampling uniformly with replacement L colors from F (in Kelsey-Kohno’s case $L = 2^{\frac{n-k+1}{2}}$). Finally, $(v, u) \in E \iff F_v \cap F_u \neq \emptyset$. They achieve from this model the same results as from the $G(n, p)$ model.

⁵ Although usually we are looking for collisions, this requirement about the cardinality of $H_{k,0}$ is needed for their analysis. Later, by our method, we will show how to use such collisions. If there are collisions, they replace the appropriate message blocks one by one until the required cardinality is obtained.

initialize a pairing set, denoted by B_k , as an empty set. This set contains pairs of the form (h_i, D_i) where h_i is a chaining value, and D_i is a message block such that $f(h_i, D_i)$ will be in the next layer of the diamond structure.

For the algorithm they define the following integers:

Let $r \geq 2, n$ be positive integers. Define the integers $s_{r,0}, s_{r,1}, \dots, s_{r,2^{r-2}}$ as follow:

$$s_{r,0} = \left\lceil 2^{\frac{n-r}{2}-1} \right\rceil, s_{r,j+1} = s_{r,j} + \left\lceil \frac{2^{\frac{n-r}{2}+1}}{2^r - 2j} \right\rceil, j = 0, 1, \dots, 2^{r-2} - 1$$

They prove that:

$$\forall j \in \{0, 1, \dots, 2^{r-2}\} : s_{r,j} \geq \frac{2^{\frac{n-r}{2}-1}}{2^r - 2j}$$

Let $i \in \{k, k-1, \dots, 3, 2\}, j \in \{0, 1, \dots, 2^{i-2} - 1\}$, The input for the step j in the phase i , denoted by $S(i, j)$, is the set $A_{i,j}$ of the $2^i - 2j$ unmatched vertices, the set $M_{i,j}$ of the $s_{i,j} \geq \frac{2^{\frac{n+i}{2}-1}}{2^i - 2j}$ message blocks generated until now, and the set $H_{i,j} = f(A_{i,j}, M_{i,j}) = \{f(a, m) | a \in A_{i,j}, m \in M_{i,j}\}$, such that

$$|H_{i,j}| = |A_{i,j}| \cdot |M_{i,j}| = (2^i - 2j) \cdot s_{i,j} \geq (2^i - 2j) \cdot \frac{2^{\frac{n+i}{2}-1}}{2^i - 2j} = 2^{\frac{n+i}{2}-1}$$

Now, they create a set $M'_{i,j}$ of $s_{i,j+1} - s_{i,j}$ message blocks such that $|f(A_{i,j}, M'_{i,j})| \geq 2^{\frac{n-i}{2}+1}$. They look for a collision, i.e., $h_{ij}, h'_{ij} \in A_{i,j}, m_{ij} \in M_{i,j}, m'_{ij} \in M'_{i,j}$ such that $f(h_{ij}, m_{ij}) = f(h'_{ij}, m'_{ij})$. Note, that since $|H_{i,j} \times f(A_{i,j}, M'_{i,j})| \geq 2^n$ the expected number of collisions is at least one, and they assume that it is exactly one.⁶ Let $A_{i,j+1} := A_{i,j} \setminus \{h_{ij}, h'_{ij}\}, M_{i,j+1} := M_{i,j} \cup M'_{i,j}$, and $H_{i,j+1} := f(A_{i,j+1}, M_{i,j+1})$. In addition set up $B_k = B_k \cup \{(h_{ij}, m_{ij}), (h'_{ij}, m'_{ij})\}$. These sets are the output of this step, and the input for the next step. The pseudo-code for this algorithm is given in Algorithm 1.

They concluded that the total message complexity of the whole diamond construction is

$$a \cdot \left(\sum_{i=2}^k 2 \cdot 2^{\frac{n+i}{2}} + 4 \cdot 2^{\frac{n}{2}} \right) \leq a \cdot 2^{\frac{n+k}{2}+2}$$

for $a = \frac{1}{4} \cdot \left[1 + \frac{1}{\sqrt{2}} + 2 \frac{e}{e-1} \left(1 + \frac{1}{\sqrt{2}} \right)^2 \right] \approx 2.732$ (the detailed analysis is in [20]).⁷

5 Our New Method

In Sect. 4.3 we discussed the time complexity when all the messages are generated simultaneously and uniformly between all vertices. If we generate about $2^{\frac{n-k+1}{2}}$

⁶ If not, they replace some message blocks one by one until it is obtained.

⁷ We note that the analysis of [20] uses a slightly different definition of a , but for more natural comparison with previous methods, we took a as the coefficient of $2^{\frac{n+k}{2}+2}$.

Algorithm 1. Kortelainen-Kortelainen's method

```

1: Input:  $H_k \subseteq 0, 1^n, |H_k| = 2^k$ 
2:  $A_{k,0} \leftarrow H_k$ 
3: Create a set  $M_{k,0}$  of  $2^{\frac{n-k}{2}-1}$  message blocks s.t.  $|f(A_{k,0}, M_{k,0})| = 2^{\frac{n-k}{2}-1}$ . (Initial-
   tialization part)
4:  $H_{k,0} \leftarrow f(A_{k,0}, M_{k,0})$ 
5:  $B_k \leftarrow \phi$ 
6: for  $i = k$  downto 2 do
7:   for  $j = 0$  to  $2^{i-2} - 1$  do
8:     Create a set  $M'_{i,j}$  of  $s_{i,j+1} - s_{i,j}$  message blocks s.t.  $M_{i,j} \cap M'_{i,j} = \phi$  and
        $|f(A_{i,j}, M'_{i,j})| \geq 2^{\frac{n-i}{2}+1}$ .
9:     look for a collision:  $h_{i,j}, h'_{i,j} \in A_{i,j}, m_{i,j} \in M_{i,j}, m'_{i,j} \in M'_{i,j}$  s.t.
        $f(h_{i,j}, m_{i,j}) = f(h'_{i,j}, m'_{i,j})$ .
10:     $A_{i,j+1} \leftarrow A_{i,j} \setminus \{h_{i,j}, h'_{i,j}\}$ 
11:     $M_{i,j+1} \leftarrow M_{i,j} \cup M'_{i,j}$ 
12:     $H_{i,j+1} \leftarrow f(A_{i,j+1}, M_{i,j+1})$ 
13:     $B_k \leftarrow B_k \cup \{(h_{i,j}, m_{i,j}), (h'_{i,j}, m'_{i,j})\}$ 
14:   end for
15: end for

```

message blocks per vertex, the expected number of collisions for a vertex is 2. In this case the $G(2^k, 2^{-k+1})$ random graph contains some isolated vertices (about $2^k \cdot e^{-2}$), and thus it does not contain any perfect matching. The conclusion of Blackburn et al. [6] is that about $\sqrt{k} \cdot 2^{\frac{n-k}{2}}$ message blocks per vertex are needed for the existence of a perfect matching in the graph. The disadvantage of this method is that we need to generate many message blocks for each vertex to get a perfect matching, of which only a small portion of the found collisions is used.

We now suggest a new method to construct the diamond. Our method is based on two ideas, described in the next sections:

1. Messages-Layers Trade-off: We generate less than $\sqrt{k} \cdot 2^{\frac{n-k}{2}}$ message blocks. Since we expect to have isolated vertices, we expect to have more than 2^{k-1} vertices in the next layer. Thus, we should add some layers to the construction to reach a single chaining value at the root of the diamond structure. We show that by generating less message blocks in each layer in exchange for more layers in the construction, we can reduce the total time complexity.
2. Match While Generate (MWG): We generate the message blocks one by one and look for collision after every generation. If two vertices collide with each other, we match them and do not generate more message blocks for them. We show that by this method we can further reduce the time complexity.

After their description, we show how to use them together to obtain the best time complexity for constructing the diamond structure.

5.1 Messages-Layers Trade-Off

The first idea, which we call “Messages-Layers Trade-off” is that we can weaken the requirement of a perfect matching in each layer exchange for more layer in the construction of the diamond. Instead of generating $\sqrt{k} \cdot 2^{\frac{n-k}{2}}$ message blocks per vertex, we generate fewer messages. The result is that in the $G(2^k, p)$ graph we get that $p < \frac{\ln n}{n}$, and thus it does not contain a perfect matching. We then match all vertices we can (e.g., by a greedy algorithm like Karp-Sipser [16] or its variants [2]) and for the remaining vertices we choose an arbitrary message block to get an arbitrary chaining value in the next layer. Since G does not contain a perfect matching, the number of vertices in the second layer is greater than 2^{k-1} . Similarly, the number of vertices in any layer $1 \leq i \leq k$ is greater than 2^{k-i} . Thus, we need to add some layers to get a single chaining value at the end of this process. We note that our experiments show that the number of layers does not increase by much. Moreover, the additional layers have almost no affect on attacks on Merkle-Damgård hash functions, and have a small impact on dithered hash functions.

We tested this idea on the Kelsey-Kohno’s case, i.e., when we generate about $2^{\frac{n-k+1}{2}}$ message blocks per vertex. In this case we get the $G(2^k, 2^{-k+1})$ model, and we know that the degree of each vertex follows $Poi(2)$ distribution. According to the handshaking lemma we know that $\sum_{i=1}^{|V|} deg(v_i) = 2|E|$, where E is the edges’ set. We also know that if $X_1, \dots, X_t \sim Poi(\lambda)$ then $\sum_{i=1}^t X_i \sim Poi(t \cdot \lambda)$. Thus, in our graph we get $|E| \sim Poi(|V|)$.⁸ We generated such a graph $G = (V, E)$ where $|V| = 2^k$ and the edges’ set E determined by sampling the $|E|$ according to $Poi(|V|)$, and for each edge sampling its two vertices uniformly between all vertices. Now we match the vertices to each other according to their degrees from low to high as follow: Let $M = \phi$ be an empty set. We run over the vertices and if $deg(v) = 1$ we add v and its single connected vertex u to the matching, i.e., $M = M \cup \{v, u\}$. In addition, we remove the vertices (and all their edges) from the graph, i.e., $G = G \setminus \{u, v\}$. We repeat it until the graph has no edges. Now we move on to the next layer with $2^k - \frac{|M|}{2}$ vertices. Clearly, this method has no advantage when $|V|$ is quite small. Thus, for the sake of simplicity, we repeat this method until $|V| \leq 16$ and then we use the Blackburn et al. [6] computation.⁹

We tested this algorithm on some different parameters for k and n . We performed 100 experiments for each pair (k, n) . The output of each experiment is the number of message blocks required to construct a diamond structure with 2^k leaves, using a compression function of n bits, and the diamond’s length.

⁸ We tested this idea on more cases: when $|E| \sim Poi(a \cdot |V|)$, $a = 0.5 + \frac{t}{20}, \forall t \in \{0, 1, 2, \dots, 19\}$. The difference between the results in the Kelsey-Kohno’s case and the best results is quite small (less than one standard deviation).

⁹ It is easy to see that if we switch to the Blackburn et al. process earlier, the expected length of the diamond will decrease.

Table 1. The number of required message blocks (represented by their \log_2), and the diamond’s length, using the Messages-Layers Trade-off method.

$n \setminus k$		14	16	18	20	22	
28	Blocks	Average	$2^{23.681}$	$2^{24.683}$	$2^{25.683}$	$2^{26.682}$	$2^{27.683}$
		S.D.	$2^{16.374}$	$2^{16.395}$	$2^{16.413}$	$2^{16.418}$	$2^{16.566}$
		Min	$2^{23.662}$	$2^{24.668}$	$2^{25.677}$	$2^{26.68}$	$2^{27.68}$
		Max	$2^{23.717}$	$2^{24.695}$	$2^{25.689}$	$2^{26.685}$	$2^{27.685}$
	Length	Average	20.13	22.85	25.8	28.34	31.35
		S.D.	1.522	1.41	1.583	1.32	1.424
		Min	18	21	24	26	30
		Max	26	31	32	34	36
32	Blocks	Average	$2^{25.683}$	$2^{26.683}$	$2^{27.682}$	$2^{28.683}$	$2^{29.683}$
		S.D.	$2^{18.117}$	$2^{18.267}$	$2^{18.553}$	$2^{18.237}$	$2^{18.541}$
		Min	$2^{25.666}$	$2^{26.665}$	$2^{27.677}$	$2^{28.679}$	$2^{29.681}$
		Max	$2^{25.709}$	$2^{26.692}$	$2^{27.689}$	$2^{28.686}$	$2^{29.684}$
	Length	Average	20.39	22.8	25.81	28.67	31.39
		S.D.	1.984	1.443	2.043	1.735	1.435
		Min	18	21	24	26	29
		Max	29	27	35	37	36
36	Blocks	Average	$2^{27.682}$	$2^{28.682}$	$2^{29.683}$	$2^{30.683}$	$2^{31.683}$
		S.D.	$2^{20.126}$	$2^{20.262}$	$2^{20.455}$	$2^{20.411}$	$2^{20.51}$
		Min	$2^{27.661}$	$2^{28.667}$	$2^{29.677}$	$2^{30.68}$	$2^{31.681}$
		Max	$2^{27.697}$	$2^{28.694}$	$2^{29.689}$	$2^{30.685}$	$2^{31.685}$
	Length	Average	20.24	22.76	25.68	28.62	31.43
		S.D.	1.525	1.443	1.723	1.523	1.416
		Min	18	21	24	27	29
		Max	27	30	33	34	36

We present here the average of these outputs from the 100 experiments and the sample standard deviation. Table 1 lists the number of message blocks (\log_2 of them) and the diamond’s length.

5.2 Match While Generate (MWG)

Intuitive Explanation of the Idea. As we discussed earlier, when we generate all the message blocks and then look for collisions, about $\sqrt{k} \cdot 2^{\frac{n+k}{2}}$ message blocks are required such that the $G(2^k, p)$ graph contains a perfect matching. In this case the expected number of collisions for each vertex is k , but in the matching we use only one of them. Our second idea, named “Match While Generate (MWG)”, is to look for collisions throughout the process, i.e., after we generate a message block M_j for v_i we check if the candidate $f(v_i, M_j)$ is already generated

by another vertex u . If yes, we match them and do not generate any additional message blocks for them. Our second idea is inspired by Hoch's thesis [14]. The process is as follow:

We keep in a Boolean array, denoted by $isMatched$, the state of each vertex, i.e., $isMatched[i] = True \iff v_i$ is already matched. The set of the chaining values for the next level is denoted by $nextV$. We start with an empty set $Candidates \subseteq \{0, 1\}^n \times V$ where $\{0, 1\}^n$ is the set of all available chaining values, and V is the set of all vertices (initial values). In each iteration j we generate a new message block M_j , and run over the vertices to calculate the chaining value candidate $h_{candidate} = f(v_i, M_j)$ for each vertex v_i . If $h_{candidate}$ is already generated by another vertex, i.e., $\exists 0 \leq r \neq i \leq 2^k - 1 : (h_{candidate}, v_r) \in Candidates$, then match v_i and v_r , add $h_{candidate}$ to $nextV$, and do not generate any additional message blocks for them. Otherwise, add $(h_{candidate}, v_i)$ to $Candidates$.

Algorithm 2 presents the Match While Generate algorithm.

Algorithm 2. Match While Generate (MWG)

```

1:  $Candidates \leftarrow \phi$ 
2:  $nextV \leftarrow \phi$ 
3: for  $i = 0$  to  $2^k - 1$  do
4:    $isMatched[i] \leftarrow False$ 
5: end for
6:  $j \leftarrow 0$ 
7:  $n_{matched} \leftarrow 0$ 
8: while  $n_{matched} < 2^k$  do
9:   Generate a message block  $M_j$ 
10:  for  $i = 0$  to  $2^k - 1$  do
11:    if  $isMatched[i]$  then
12:      Go to 10
13:    end if
14:    Calculate  $h_{candidate} = f(v_i, M_j)$ 
15:    if  $\exists 0 \leq r \leq 2^k - 1, r \neq i : (h_{candidate}, v_r) \in Candidates^a \wedge \neg isMatched[r]$ 
16:      then
17:         $nextV \leftarrow nextV \cup \{h_{candidate}\}$ 
18:         $isMatched[i] \leftarrow True$ 
19:         $isMatched[r] \leftarrow True$ 
20:         $n_{matched} \leftarrow n_{matched} + 2$ 
21:      else
22:         $Candidates \leftarrow Candidates \cup \{(h_{candidate}, v_i)\}$ 
23:      end if
24:    end for
25:   $j \leftarrow j + 1$ 
26: end while

```

^a We can use a hash table to keep the $Candidates$ elements, to maintain a constant search time.

Analysis of the Algorithm. Here we suggest a recursive presentation for the diamond structure construction. We present here the construction of the first layer of the diamond structure (the application for the other layers is immediate). For the reading convenience we fix an arbitrary order of the vertices v_0, \dots, v_{2^k-1} . Let us define some random variables, depending on the process “time” t , i.e., the generation of the t 'th candidate:

- $m(t)$:= The number of matched vertices at time t .
- $c(t)$:= The number of candidates available for matching at time t .
- $bu(t)$:= The number of candidates generated from the current vertex at time t , i.e., sum of values in *Candidates* of the form (h_i, v) .
- $be(t)$:= The number of unmatched vertices before the current vertex at time t .
- $af(t)$:= The number of unmatched vertices after the current vertex at time t .
- We also use a counter variable to count the number of self-collisions, denoted by sc .

The initialization of the variables is: $sc = m(0) = c(0) = bu(0) = be(0) = 0$, and $af(0) = 2^k - 1$. At time $t + 1$ we generate a new candidate, and we compute the value of all the variables, given the values of all variables at time t . When we generate a new candidate from a vertex v_i , there are four possible cases:

1. The new candidate leads to a self-collision. In this case we continue to generate new candidates from the current vertex until it is not a self-collision. For each new message block, we increment the sc counter by one. This case happens with probability $\frac{bu(t)}{2^n}$.
2. The new candidate leads to a matching with v_j , for $j < i$. We have a matching, and since $j < i$ we match a vertex whose position is before the current position. In addition, we remove these two vertices, where v_i with $bu(t)$ candidates, and v_j with $bu(t) + 1$ candidates. Finally, we move to the next vertex. Thus, the updating of the variables is as follow: $m(t + 1) = m(t) + 2$, $be(t + 1) = be(t) - 1$, $af(t + 1) = af(t) - 1$ and $c(t + 1) = c(t) - 2 \cdot bu(t) - 1$. This case happens with probability $\frac{be(t) \cdot (bu(t) + 1)}{2^n}$.
3. The new candidate leads to a matching with v_j , for $j > i$. We have a matching, and since $j > i$, we match a vertex whose position is after the current position. In addition, we remove these two vertices, both with $bu(t)$ candidates. Finally, we move to the next vertex. Thus, the updating of the variables is as follow: $m(t + 1) = m(t) + 2$, $af(t + 1) = af(t) - 2$, $be(t + 1) = be(t)$ and $c(t + 1) = c(t) - 2 \cdot bu(t)$. This case happens with probability $\frac{af(t) \cdot bu(t)}{2^n}$.
4. The new candidate does not lead to any matching. There is no matching, and we add the new candidate. In addition, we move to the next vertex, so that the current vertex is added to the vertices that before the next, and the next removed from the vertices that are after it. Thus, the updating of the variables is as follow: $m(t + 1) = m(t)$, $be(t + 1) = be(t) + 1$, $af(t + 1) = af(t) - 1$ and $c(t + 1) = c(t) + 1$. This case happens with probability $1 - \frac{bu(t)}{2^n} - \frac{be(t) \cdot (bu(t) + 1)}{2^n} - \frac{af(t) \cdot bu(t)}{2^n}$.

Finally, if $af(t + 1) < 0$ (after the above updates), it means that we finished an iteration over the unmatched vertices, and at time $t + 1$ we start a new iteration.

Thus, $be(t + 1) = 0, af(t + 1) = 2^k - m(t + 1) - 1$ and $bu(t + 1) = bu(t) + 1$. If $af(t + 1) \geq 0$ it means that we remain at the same iteration and thus $bu(t + 1) = bu(t)$.

Using this recursion, we tested the MWG algorithm with some parameters for k and n . We performed 100 experiments for each pair (k, n) . The output of each experiment is the number of message blocks required to construct a diamond structure with 2^k leaves, using a compression function of n bits. We present in Table 2 the average of these numbers from the 100 experiments and the sample standard deviation.

Table 2. The number of required message blocks (represented by their \log_2), using the MWG algorithm.

$n \setminus k$		14	16	18	20	22
28	Average	$2^{23.399}$	$2^{24.411}$	$2^{25.418}$	$2^{26.423}$	$2^{27.43}$
	S.D.	$2^{16.175}$	$2^{16.289}$	$2^{16.65}$	$2^{16.67}$	$2^{16.666}$
	Min	$2^{23.374}$	$2^{24.396}$	$2^{25.411}$	$2^{26.42}$	$2^{27.428}$
	Max	$2^{23.425}$	$2^{24.423}$	$2^{25.428}$	$2^{26.428}$	$2^{27.431}$
32	Average	$2^{25.4}$	$2^{26.411}$	$2^{27.417}$	$2^{28.421}$	$2^{29.423}$
	S.D.	$2^{18.245}$	$2^{18.683}$	$2^{18.839}$	$2^{18.795}$	$2^{18.881}$
	Min	$2^{25.375}$	$2^{26.389}$	$2^{27.403}$	$2^{28.417}$	$2^{29.421}$
	Max	$2^{25.429}$	$2^{26.427}$	$2^{27.426}$	$2^{28.424}$	$2^{29.425}$
36	Average	$2^{27.399}$	$2^{28.41}$	$2^{29.416}$	$2^{30.419}$	$2^{31.422}$
	S.D.	$2^{20.237}$	$2^{20.465}$	$2^{20.831}$	$2^{20.788}$	$2^{20.975}$
	Min	$2^{27.371}$	$2^{28.397}$	$2^{29.406}$	$2^{30.416}$	$2^{31.419}$
	Max	$2^{27.425}$	$2^{28.429}$	$2^{29.426}$	$2^{30.423}$	$2^{31.424}$

Actual Experiments. In addition to the simulations which are based on the mathematical model, we tested this algorithm on a diamond structure with 2^{18} leaves, using a 28-bit compression function (we used the 28 first bits of SHA1), i.e., $k = 18, n = 28$. According to Kelsey and Kohno [17] we should generate about $2^{\frac{28+18}{2}+2} = 2^{25}$ message blocks. Blackburn et al. [6] prove that by Kelsey-Kohno’s method about $\sqrt{18} \cdot 2^{\frac{28+18}{2}+2} > 2^{27}$ message blocks are needed. According to Kortelainen and Kortelainen we should generate about $a \cdot 2^{\frac{28+18}{2}+2} > 2^{26}$ message blocks. We constructed a diamond structure 100 times with different initial values. The mean value of the number of required message blocks was $\mu = 45672583.18 \approx 2^{25.445}$, and the sample standard deviation was $\sigma = 104866.202 \approx 2^{16.678}$. Figure 3 illustrates the distribution of the number of message blocks required to construct it (the numbers are represented by their \log_2). According to the t -Test, the data follows $Norm(\mu, \sigma^2)$ distribution, with statistical significance of $2.5091 \cdot 10^{-14}$.¹⁰

¹⁰ We note that the mean value of the experiment is greater than those of the recursion by a few standard deviation units. This difference is a subject for a future research.

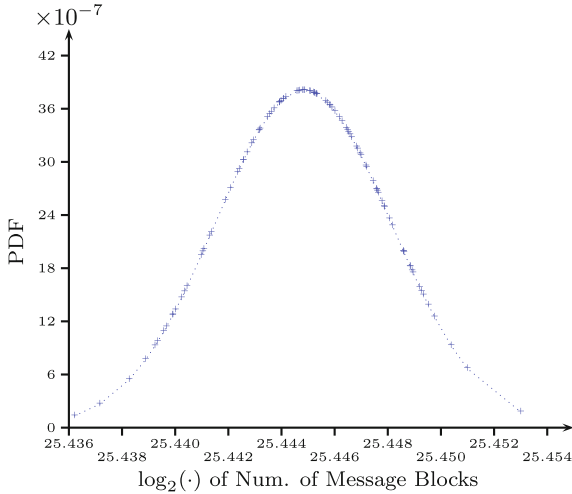


Fig. 3. Sampled distribution of the number of message blocks (represented by its \log_2) required to construct a diamond structure with 2^{18} leaves, and compression function of 28 bits, using the MWG algorithm.

5.3 Combining These Two Ideas Together

It is possible to improve the MWG algorithm, using the method described in Sect. 5.1. Before we present the improvement, we want to look at the number of message blocks required for each collision, throughout the construction of a diamond structure’s first layer. Intuitively, at the beginning of the construction there are only a few candidates for matching, and thus we need to generate many message blocks for a collision. Similarly, near the end of the layer, since only a few unmatched vertices remain, i.e., only a few candidates for matching exist, we need to generate many message blocks for a collision. In the middle part of the layer, on the one hand we already generated a significant amount of message blocks per vertex, and on the other hand still have a significant amount of unmatched vertices, so we have enough candidates, thus, we expect a collision after fewer messages. We tested this intuition, and Fig. 4 shows an example for the average number of message blocks generated between 256 collisions using the MWG algorithm, where $k = 18$, $n = 28$.

As we discussed above, the investment of the beginning is needed to ensure we have enough candidates. At the same time, the last matchings require a lot of message blocks in return of a small benefit. Thus, we can stop the construction near the end, as was discussed in Sect. 5.1, and move to the next layer. As a result we will have to add some layers. To do so, we improve the MWG algorithm by adding a boundary on the number of message blocks. Clearly, this method has no advantage over the previous when $|V|$ is quite small, and it may even be the case that it is less efficient. Thus, for the sake of simplicity, we repeat this method until $|V| \leq 16$ and then we use the original MWG algorithm.

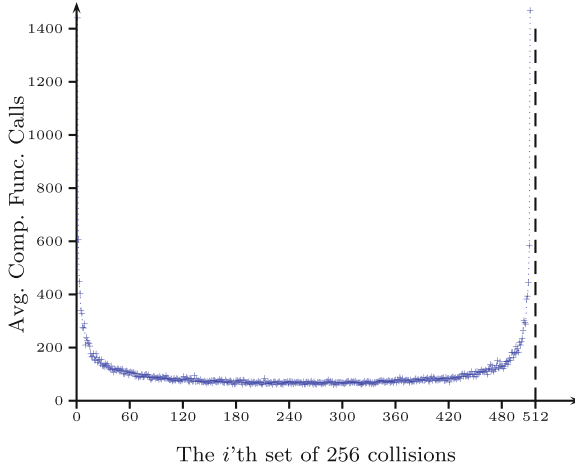


Fig. 4. Average number of compression functions calls needed for 256 collisions during the construction of the first layer of the diamond structure (for $k = 18, n = 28$)

We tested this improved algorithm by bounding the number of message blocks by $2^{\frac{n+k+1}{2}}$ (Kelsey-Kohno’s number) for $n = 28, k = 18$. We adapted the recursion presented in Sect. 5.2 to this improved algorithm, and using the adapted recursion we tested the improved MWG algorithm on some parameters for k and n . We performed 100 experiments for each pair (k, n) . The output of each experiment is the number of message blocks required to construct a diamond structure with 2^k leaves, using a compression function of n bits, and the diamond’s length. We present here the average of these outputs from the 100 experiments and the sample standard deviation. Table 3 lists the number of message blocks (\log_2 of them) and the diamond’s length.

In addition to the simulations which are based on the mathematical model, we also tested this improved algorithm on a diamond structure with 2^{18} leaves, using a 28-bit compression function (we used the 28 first bits of SHA1), i.e., $k = 18, n = 28$. We constructed a diamond structure 100 times with different initial values. The mean value of the number of required message blocks was $\mu = 42078721.93 \approx 2^{25.327}$, and the sample standard deviation was $\sigma = 58941.064 \approx 2^{15.847}$. Figure 5 illustrates the distribution of the number of message blocks required to construct it (the numbers are represented by their \log_2). According to the t -Test, the data follows $Norm(\mu, \sigma^2)$ distribution, with statistical significance of $4.0967 \cdot 10^{-14}$.¹¹ The experiments suggest that for our method of constructing diamond structures $c = 1.254$.

¹¹ We note that the mean value of the experiment is greater than those of the recursion by a few s.d. units. This difference is a subject for a future research.

Table 3. The number of required message blocks (represented by their \log_2), and the diamond’s length, using the improved MWG algorithm.

$n \setminus k$		14	16	18	20	22	
28	Blocks	Average	$2^{23.304}$	$2^{24.309}$	$2^{25.313}$	$2^{26.315}$	$2^{27.318}$
		S.D.	$2^{15.946}$	$2^{15.688}$	$2^{15.672}$	$2^{15.842}$	$2^{15.945}$
		Min	$2^{23.279}$	$2^{24.301}$	$2^{25.308}$	$2^{26.313}$	$2^{27.317}$
		Max	$2^{23.323}$	$2^{24.321}$	$2^{25.317}$	$2^{26.318}$	$2^{27.32}$
	Length	Average	15.01	17	19.01	21.07	23.11
		S.D.	0.1	0	0.1	0.256	0.314
		Min	15	17	19	21	23
		Max	16	17	20	22	24
32	Blocks	Average	$2^{25.304}$	$2^{26.309}$	$2^{27.312}$	$2^{28.314}$	$2^{29.315}$
		S.D.	$2^{17.667}$	$2^{17.627}$	$2^{17.678}$	$2^{17.776}$	$2^{17.865}$
		Min	$2^{25.282}$	$2^{26.3}$	$2^{27.306}$	$2^{28.311}$	$2^{29.314}$
		Max	$2^{25.323}$	$2^{26.318}$	$2^{27.315}$	$2^{28.317}$	$2^{29.316}$
	Length	Average	15.01	17.02	19.03	21.01	23.07
		S.D.	0.1	0.141	0.171	0.1	0.256
		Min	15	17	19	21	23
		Max	16	18	20	22	24
36	Blocks	Average	$2^{27.303}$	$2^{28.309}$	$2^{29.312}$	$2^{30.313}$	$2^{31.314}$
		S.D.	$2^{19.639}$	$2^{19.893}$	$2^{19.756}$	$2^{19.67}$	$2^{19.765}$
		Min	$2^{27.287}$	$2^{28.296}$	$2^{29.308}$	$2^{30.311}$	$2^{31.313}$
		Max	$2^{27.322}$	$2^{28.319}$	$2^{29.317}$	$2^{30.315}$	$2^{31.315}$
	Length	Average	15	17.03	19.02	21.02	23.1
		S.D.	0	0.171	0.141	0.141	0.302
		Min	15	17	19	21	23
		Max	15	18	20	22	24

6 Summary

In this paper we showed a time complexity optimization for the construction of *diamond structures*. We presented two ideas to optimize the construction:

1. Messages-Layers Trade-off: We generate less message blocks in each layer in exchange for more layers in the construction.
2. Match While Generate (MWG): We generate the message blocks one by one and look for collision after every generation.

We also showed how to combine these two ideas together to improve the MWG algorithm. Using the improved MWG we got the best results to date with respect to time complexity.

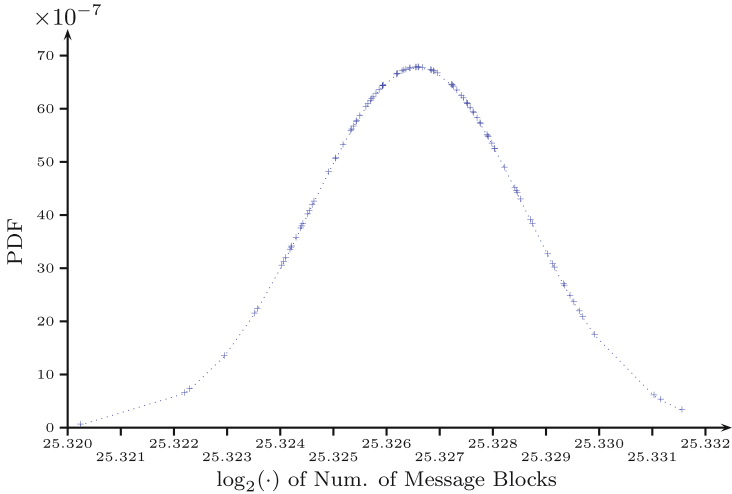


Fig. 5. Sampled distribution of the number of message blocks (represented by its \log_2) required to construct a diamond structure with 2^{18} leaves, and compression function of 28 bits, using the improved MWG algorithm.

For comparison, we present in Table 4 the number of required message blocks using the previous methods and using our improved MWG algorithm, for each (k, n) .

Table 4. Comparing the time complexity of the different methods.

Method	Time complexity
Kelsey-Kohno [17] ^a	$2^{\frac{n+k}{2}+2}$
Blackburn et al. [6]	$\sqrt{k} \cdot 2^{\frac{n+k}{2}+2}$
Kortelainen-Kortelainen [21]	$\frac{1 + \frac{1}{\sqrt{2}} + 2 \frac{e}{e-1} \left(1 + \frac{1}{\sqrt{2}}\right)^2}{4} \cdot 2^{\frac{n+k}{2}+2} \approx 2.732 \cdot 2^{\frac{n+k}{2}+2}$
Improved MWG	$1.254 \cdot 2^{\frac{n+k}{2}+2}$

^a We remind the reader that Kelsey-Kohno’s analysis is inaccurate.







Acknowledgements. The research of Ariel Weizmann was supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister’s Office. The second author was supported in part by the Israeli Science Foundation through grant No. 827/12.

References

1. Andreeva, E., Bouillaguet, C., Dunkelman, O., Fouque, P., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: New second-preimage attacks on hash functions. *J. Cryptol.* **29**(4), 657–696 (2016)
2. Aronson, J., Frieze, A., Pittel, B.G.: Maximum matchings in sparse random graphs: Karp-Sipser revisited. *Random Struct. Algorithms* **12**, 111–178 (1998)
3. Barham, M., Dunkelman, O., Lucks, S., Stevens, M.: New second preimage attacks on dithered hash functions with low memory complexity. In: Avanzi, R., Heys, H. (eds.) *Selected Areas in Cryptography – SAC 2016*. LNCS, vol. 10532, pp. 247–263. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_14
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. *Submiss. NIST (Round 2)* **3**, 30 (2009)
5. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and reduced SHA-1. In: Cramer [8], pp. 36–57
6. Blackburn, S.R., Stinson, D.R., Upadhyay, J.: On the complexity of the herding attack and some related attacks on hash functions. *Des. Codes Crypt.* **64**(1–2), 171–193 (2012)
7. Brassard, G. (ed.): *CRYPTO 1989*. LNCS, vol. 435. Springer, New York (1990). <https://doi.org/10.1007/0-387-34805-0>
8. Cramer, R. (ed.): *EUROCRYPT 2005*. LNCS, vol. 3494. Springer, Heidelberg (2005). <https://doi.org/10.1007/b136415>
9. Damgård, I.: A design principle for hash functions. In: Brassard [7], pp. 416–427
10. Dean, R.D.: Formal aspects of mobile code security. Ph.D. thesis, Princeton University, Princeton (1999)
11. Erdős, P., Rényi, A.: On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* **5**, 17–61 (1960)
12. Erdős, P., Rényi, A.: On the strength of connectedness of a random graph. *Acta Math. Hung.* **12**(1–2), 261–267 (1961)
13. Erdős, P., Rényi, A.: On the existence of a factor of degree one of a connected random graph. *Acta Math. Hung.* **17**(3–4), 359–368 (1966)
14. Hoch, Y.Z.: Security analysis of generic iterated hash functions. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel (2009)
15. Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In: Franklin, M. (ed.) *CRYPTO 2004*. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_19
16. Karp, R.M., Sipser, M.: Maximum matchings in sparse random graphs. In: *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28–30 October 1981*, pp. 364–375. IEEE Computer Society (1981)
17. Kelsey, J., Kohno, T.: Herding hash functions and the Nostradamus attack. In: Vaudenay, S. (ed.) *EUROCRYPT 2006*. LNCS, vol. 4004, pp. 183–200. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_12
18. Kelsey, J., Schneier, B.: Second preimages on n -bit hash functions for much less than 2^n work. In: Cramer [8], pp. 474–490
19. Klima, V.: Finding MD5 collisions on a notebook PC using multi-message modifications. *Cryptology ePrint Archive, Report 2005/102* (2005)
20. Kortelainen, T.: On iteration-based security flaws in modern hash functions. Ph.D. thesis, University of Oulu, Finland (2014)
21. Kortelainen, T., Kortelainen, J.: On diamond structures and Trojan message attacks. In: Sako, K., Sarkar, P. (eds.) *ASIACRYPT 2013*. LNCS,

- vol. 8270, pp. 524–539. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_27
22. Merkle, R.C.: One way hash functions and DES. In: Brassard [7], pp. 428–446
 23. Rivest, R.L.: Abelian square-free dithering for iterated hash functions. Presented at ECRYPT hash function workshop, Cracow, 21 June 2005, and at the cryptographic hash workshop, Gaithersburg, Maryland, 1 November 2005, August 2005
 24. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_8
 25. Stevens, M.: Attacks on hash functions and applications. Ph.D. thesis, Leiden University (2012)
 26. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_4
 27. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer [8], pp. 1–18
 28. 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
 29. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer [8], pp. 19–35

Efficient Optimal Ate Pairing at 128-Bit Security Level

Md. Al-Amin Khandaker¹ , Yuki Nanjo¹ , Loubna Ghammam² ,
Sylvain Duquesne³ , Yasuyuki Nogami¹ , and Yuta Kodera¹ 

¹ Faculty of Engineering, Okayama University, Okayama 7008530, Japan
{khandaker, yuki.nanjo, yuta.kodera}@s.okayama-u.ac.jp,

yasuyuki.nogami@okayama-u.ac.jp
² Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC,
14000 Caen, France
ghammam.loubna@yahoo.fr

³ Univ Rennes, CNRS, IRMAR - UMR 6625, 35000 Rennes, France
sylvain.duquesne@univ-rennes1.fr

Abstract. Following the emergence of Kim and Barbulescu’s new number field sieve (exTNFS) algorithm at CRYPTO’16 [21] for solving discrete logarithm problem (DLP) over the finite field; pairing-based cryptography researchers are intrigued to find new parameters that confirm standard security levels against exTNFS. Recently, Barbulescu and Duquesne have suggested new parameters [3] for well-studied pairing-friendly curves i.e., Barreto-Naehrig (BN) [5], Barreto-Lynn-Scott (BLS-12) [4] and Kachisa-Schaefer-Scott (KSS-16) [19] curves at 128-bit security level (twist and sub-group attack secure). They have also concluded that in the context of Optimal-Ate pairing with their suggested parameters, BLS-12 and KSS-16 curves are more efficient choices than BN curves. Therefore, this paper selects the atypical and less studied pairing-friendly curve in literature, i.e., KSS-16 which offers quartic twist, while BN and BLS-12 curves have sextic twist. In this paper, the authors optimize Miller’s algorithm of Optimal-Ate pairing for the KSS-16 curve by deriving efficient sparse multiplication and implement them. Furthermore, this paper concentrates on the Miller’s algorithm to experimentally verify Barbulescu et al.’s estimation. The result shows that Miller’s algorithm time with the derived pseudo 8-sparse multiplication is most efficient for KSS-16 than other two curves. Therefore, this paper defends Barbulescu and Duquesne’s conclusion for 128-bit security.

Keywords: KSS-16 curve · Optimal-Ate pairing · Sparse multiplication

1 Introduction

Since the inception by Sakai et al. [25], pairing-based cryptography has gained much attention to cryptographic researchers as well as to mathematicians. It gives flexibility to protocol researcher to innovate applications with provable

security and at the same time to mathematicians and cryptography engineers to find efficient algorithms to make pairing implementation more efficient and practical. This paper tries to efficiently carry out the basic operation of a specific type of pairing calculation over certain pairing-friendly curves.

Generally, a pairing is a bilinear map e typically defined as $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where \mathbb{G}_1 and \mathbb{G}_2 are additive cyclic sub-groups of order r on a certain elliptic curve E over a finite extension field \mathbb{F}_{p^k} and \mathbb{G}_T is a multiplicative cyclic group of order r in $\mathbb{F}_{p^k}^*$. Let $E(\mathbb{F}_p)$ be the set of rational points over the prime field \mathbb{F}_p which forms an additive Abelian group together with the point at infinity \mathcal{O} . The total number of rational points is denoted as $\#E(\mathbb{F}_p)$. Here, the order r is a large prime number such that $r \mid \#E(\mathbb{F}_p)$ and $\gcd(r, p) = 1$. The embedding degree k is the smallest positive integer such that $r \mid (p^k - 1)$. Two basic properties of pairing are

- bilinearity is such that $\forall P_i \in \mathbb{G}_1$ and $\forall Q_i \in \mathbb{G}_2$, where $i = 1, 2$, then $e(Q_1 + Q_2, P_1) = e(Q_1, P_1) \cdot e(Q_2, P_1)$ and $e(Q_1, P_1 + P_2) = e(Q_1, P_1) \cdot e(Q_1, P_2)$,
- and e is non-degenerate means $\forall P \in \mathbb{G}_1$ there is a $Q \in \mathbb{G}_2$ such that $e(Q, P) \neq 1$ and $\forall Q \in \mathbb{G}_2$ there is a $P \in \mathbb{G}_1$ such that $e(P, Q) \neq 1$.

Such properties allows researchers to come up with various cryptographic applications including ID-based encryption [8], group signature authentication [7], and functional encryption [24]. However, the security of pairing-based cryptosystems depends on

- the difficulty of solving elliptic curve discrete logarithm problem (ECDLP) in the groups of order r over \mathbb{F}_p ,
- the infeasibility of solving the discrete logarithm problem (DLP) in the multiplicative group $\mathbb{G}_T \in \mathbb{F}_{p^k}^*$,
- and the difficulty of pairing inversion.

To maintain the same security level in both groups, the size of the order r and extension field p^k is chosen accordingly. If the desired security level is δ then $\log_2 r \geq 2\delta$ is desirable due to Pollard's rho algorithm. For efficient pairing, the ratio $\rho = \log_2 p^k / \log_2 r \approx 1$, is expected (usually $1 \leq \rho \leq 2$). In practice, elliptic curves with small embedding degrees k and large r are selected and commonly are known as "pairing-friendly" elliptic curves.

Galbraith et al. [15] have classified pairings as three major categories based on the underlying group's structure as

- Type 1, where $\mathbb{G}_1 = \mathbb{G}_2$, also known as symmetric pairing.
- Type 2, where $\mathbb{G}_1 \neq \mathbb{G}_2$, known as asymmetric pairing. There exists an efficiently computable isomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$ but none in reverse direction.
- Type 3, which is also asymmetric pairing, i.e., $\mathbb{G}_1 \neq \mathbb{G}_2$. But no efficiently computable isomorphism is known in either direction between \mathbb{G}_1 and \mathbb{G}_2 .

This paper chooses one of the Type 3 variants of pairing named as Optimal-Ate [29] with Kachisa-Schaefer-Scott (KSS) [19] pairing-friendly curve of embedding degree $k = 16$. Few previous works have been done on this curve. Zhang et al. [31]

have shown the computational estimation of the Miller’s loop and proposed efficient final exponentiation for 192-bit security level in the context of Optimal-Ate pairing over KSS-16 curve. A few years later Ghammam et al. [16] have shown that KSS-16 is the best suited for multi-pairing (i.e., the product and/or the quotient) when the number of pairing is more than two. Ghammam et al. [16] also corrected the flaws of proposed final exponentiation algorithm by Zhang et al. [31] and proposed a new one and showed the vulnerability of Zhang’s parameter settings against small subgroup attack. The recent development of NFS by Kim and Barbulescu [21] requires updating the parameter selection for all the existing pairings over the well known pairing-friendly curve families such as BN [5], BLS [13] and KSS [19]. The most recent study by Barbulescu et al. [3] have shown the security estimation of the current parameter settings used in well-studied curves and proposed new parameters, resistant to small subgroup attack.

Barbulescu and Duquesne’s study finds that the current parameter settings for 128-bit security level on BN-curve studied in literature can withstand for 100-bit security. Moreover, they proposed that BLS-12 and surprisingly KSS-16 are the most efficient choice for Optimal-Ate pairing at the 128-bit security level. Therefore, the authors focus on the efficient implementation of the less studied KSS-16 curve for Optimal-Ate pairing by applying the most recent parameters. Mori et al. [23] and Khandaker et al. [20] have shown a specific type of sparse multiplication for BN and KSS-18 curve respectively where both of the curves supports sextic twist. The authors have extended the previous works for quartic twisted KSS-16 curve and derived pseudo-8 sparse multiplication for line evaluation step in the Miller’s algorithm. As a consequence, the authors made the choice to concentrate on Miller’s algorithm’s execution time and computational complexity to verify the claim of [3]. The implementation shows that Miller’s algorithm time has a tiny difference between KSS-16 and BLS-12 curves. However, they both are more efficient and faster than BN curve.

2 Fundamentals of Elliptic Curve and Pairing

2.1 Kachisa-Schaefer-Scott (KSS) Curve

In [19], Kachisa, Schaefer, and Scott proposed a family of non super-singular pairing-friendly elliptic curves of embedding degree $k = \{16, 18, 32, 36, 40\}$, using elements in the cyclotomic field. In what follows, this paper considers the curve of embedding degree $k = 16$, named as *KSS-16*, defined over extension field $\mathbb{F}_{p^{16}}$ as follows:

$$E/\mathbb{F}_{p^{16}} : Y^2 = X^3 + aX, \quad (a \in \mathbb{F}_p) \text{ and } a \neq 0, \quad (1)$$

where $X, Y \in \mathbb{F}_{p^{16}}$. Similar to other pairing-friendly curves, *characteristic* p , *Frobenius trace* t and *order* r of this curve are given by the following polynomials of integer variable u .

$$p(u) = (u^{10} + 2u^9 + 5u^8 + 48u^6 + 152u^5 + 240u^4 + 625u^2 + 2398u + 3125)/980, \quad (2a)$$

$$r(u) = (u^8 + 48u^4 + 625)/61255, \quad (2b)$$

$$t(u) = (2u^5 + 41u + 35)/35, \quad (2c)$$

where u is such that $u \equiv 25$ or $45 \pmod{70}$ and the ρ value is $\rho = (\log_2 p / \log_2 r) \approx 1.25$. The total number of rational points $\#E(\mathbb{F}_p)$ is given by Hasse's theorem as, $\#E(\mathbb{F}_p) = p + 1 - t$. When the definition field is the k -th degree extension field \mathbb{F}_{p^k} , rational points on the curve E also form an additive Abelian group denoted as $E(\mathbb{F}_{p^k})$. Total number of rational points $\#E(\mathbb{F}_{p^k})$ is given by Weil's theorem [30] as $\#E(\mathbb{F}_{p^k}) = p^k + 1 - t_k$, where $t_k = \alpha^k + \beta^k$. α and β are complex conjugate numbers.

2.2 Extension Field Arithmetic and Towering

Pairing-based cryptography requires performing the arithmetic operation in extension fields of degree $k \geq 6$ [28]. Consequently, such higher degree extension field needs to be constructed as a tower of sub-fields [6] to perform arithmetic operation cost efficiently. Bailey et al. [2] have explained optimal extension field by towering by using irreducible binomials.

Towering of $\mathbb{F}_{p^{16}}$ extension field: For KSS-16 curve, $\mathbb{F}_{p^{16}}$ construction process given as follows using tower of sub-fields.

$$\begin{cases} \mathbb{F}_{p^2} = \mathbb{F}_p[\alpha]/(\alpha^2 - c), \\ \mathbb{F}_{p^4} = \mathbb{F}_{p^2}[\beta]/(\beta^2 - \alpha), \\ \mathbb{F}_{p^8} = \mathbb{F}_{p^4}[\gamma]/(\gamma^2 - \beta), \\ \mathbb{F}_{p^{16}} = \mathbb{F}_{p^8}[\omega]/(\omega^2 - \gamma), \end{cases} \quad (3)$$

where $p \equiv 5 \pmod{8}$ and c is a quadratic non residue in \mathbb{F}_p . This paper considers $c = 2$ along with the value of the parameter u as given in [3].

Towering of $\mathbb{F}_{p^{12}}$ extension field: Let $6|(p-1)$, where p is the characteristics of BN or BLS-12 curve and -1 is a quadratic and cubic non-residue in \mathbb{F}_p since $p \equiv 3 \pmod{4}$. In the context of BN or BLS-12, where $k = 12$, $\mathbb{F}_{p^{12}}$ is constructed as a tower of sub-fields with irreducible binomials as follows:

$$\begin{cases} \mathbb{F}_{p^2} = \mathbb{F}_p[\alpha]/(\alpha^2 + 1), \\ \mathbb{F}_{p^6} = \mathbb{F}_{p^2}[\beta]/(\beta^3 - (\alpha + 1)), \\ \mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[\gamma]/(\gamma^2 - \beta). \end{cases} \quad (4)$$

Extension Field Arithmetic of $\mathbb{F}_{p^{16}}$ and $\mathbb{F}_{p^{12}}$. Among the arithmetic operations multiplication, squaring and inversion are regarded as expensive operation than addition/subtraction. The calculation cost, based on number of prime field multiplication M_p and squaring S_p is given in Table 1. The arithmetic operations in \mathbb{F}_p are denoted as M_p for a multiplication, S_p for a squaring, I_p for an inversion and m with suffix denotes multiplication with basis element. However, squaring is more optimized by using Devegili et al.'s [11] complex squaring technique which cost $2M_p + 4A_p + 2m_\alpha$ for one squaring operation in \mathbb{F}_{p^2} . In total it costs $54M_p$ for one squaring in $\mathbb{F}_{p^{16}}$. Table 1 shows the operation estimation for $\mathbb{F}_{p^{16}}$.

Table 1. Number of arithmetic operations in $\mathbb{F}_{p^{16}}$ based on Eq. (3)

$M_{p^2} = 3M_p + 5A_p + 1m_\alpha \rightarrow 3M_p$	$S_{p^2} = 3S_p + 4A_p + 1m_\alpha \rightarrow 3S_p$
$M_{p^4} = 3M_{p^2} + 5A_{p^2} + 1m_\beta \rightarrow 9M_p$	$S_{p^4} = 3S_{p^2} + 4A_{p^2} + 1m_\beta \rightarrow 9S_p$
$M_{p^8} = 3M_{p^4} + 5A_{p^4} + 1m_\gamma \rightarrow 27M_p$	$S_{p^8} = 3S_{p^4} + 4A_{p^4} + 1m_\gamma \rightarrow 27S_p$
$M_{p^{16}} = 3M_{p^8} + 5A_{p^8} + 1m_\omega \rightarrow 81M_p$	$S_{p^{16}} = 3M_{p^8} + 4A_{p^8} + 1m_\omega \rightarrow 81S_p$

Table 2 shows the operation estimation for $\mathbb{F}_{p^{12}}$ according to the tower shown in Eq. (4). The algorithms for \mathbb{F}_{p^2} and \mathbb{F}_{p^3} multiplication and squaring given in [12] have been used in this paper to construct the $\mathbb{F}_{p^{12}}$ extension field arithmetic.

Table 2. Number of arithmetic operations in $\mathbb{F}_{p^{12}}$ based on Eq. (4)

$M_{p^2} = 3M_p + 5A_p + 1m_\alpha \rightarrow 3M_p$	$S_{p^2} = 2S_p + 3A_p \rightarrow 2S_p$
$M_{p^6} = 6M_{p^2} + 15A_{p^2} + 2m_\beta \rightarrow 18M_p$	$S_{p^6} = 2M_{p^2} + 3S_{p^2} + 9A_{p^2} + 2m_\beta \rightarrow 12S_p$
$M_{p^{12}} = 3M_{p^6} + 5A_{p^6} + 1m_\gamma \rightarrow 54M_p$	$S_{p^{12}} = 2M_{p^6} + 5A_{p^6} + 2m_\gamma \rightarrow 36S_p$

2.3 Ate and Optimal-Ate on KSS-16, BN, BLS-12 Curve

A brief of pairing and its properties are described in Sect. 1. In the context of pairing on the targeted pairing-friendly curves, two additive rational point groups $\mathbb{G}_1, \mathbb{G}_2$ and a multiplicative group \mathbb{G}_T of order r are considered. $\mathbb{G}_1, \mathbb{G}_2$ and \mathbb{G}_T are defined as follows:

$$\begin{aligned}
 \mathbb{G}_1 &= E(\mathbb{F}_p)[r] \cap \text{Ker}(\pi_p - [1]), \\
 \mathbb{G}_2 &= E(\mathbb{F}_{p^k})[r] \cap \text{Ker}(\pi_p - [p]), \\
 \mathbb{G}_T &= \mathbb{F}_{p^k}^* / (\mathbb{F}_{p^k}^*)^r, \\
 e &: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T,
 \end{aligned}
 \tag{5}$$

where e denotes Ate pairing [9]. $E(\mathbb{F}_{p^k})[r]$ denotes rational points of order r and $[n]$ denotes n times scalar multiplication for a rational point. π_p denotes the Frobenius endomorphism given as $\pi_p : (x, y) \mapsto (x^p, y^p)$.

KSS-16 Curve: In what follows, we consider $P \in \mathbb{G}_1 \subset E(\mathbb{F}_p)$ and $Q \in \mathbb{G}_2 \subset E(\mathbb{F}_{p^{16}})$ for KSS-16 curves. Ate pairing $e(Q, P)$ is given as follows:

$$e(Q, P) = f_{t-1, Q}(P)^{\frac{p^{16}-1}{r}}, \quad (6)$$

where $f_{t-1, Q}(P)$ symbolizes the output of Miller's algorithm and $\lfloor \log_2(t-1) \rfloor$ is the loop length. The bilinearity of Ate pairing is satisfied after calculating the final exponentiation $(p^k - 1)/r$.

Vercauteren proposed more efficient variant of Ate pairing named as Optimal-Ate pairing [29] where the Miller's loop length reduced to $\lfloor \log_2 u \rfloor$. The previous work of Zhang et al. [31] has derived the optimal Ate pairing on the KSS-16 curve which is defined as follows with $f_{u, Q}(P)$ is the Miller function evaluated on P :

$$e_{opt}(Q, P) = ((f_{u, Q}(P) \cdot l_{\lfloor u \rfloor Q, \lfloor p \rfloor Q}(P))^{p^3} \cdot l_{Q, Q}(P))^{\frac{p^{16}-1}{r}}. \quad (7)$$

The formulas for Optimal-Ate pairing for the target curves are given in Table 3.

Table 3. Optimal Ate pairing formulas for target curves

Curve	Miller's Algo.	Final Exp.
KSS-16	$(f_{u, Q}(P) \cdot l_{\lfloor u \rfloor Q, \lfloor p \rfloor Q}(P))^{p^3} \cdot l_{Q, Q}(P)$	$(p^{16} - 1)/r$
BN	$f_{6u+2, Q}(P) \cdot l_{\lfloor 6u+2 \rfloor Q, \lfloor p \rfloor Q}(P) \cdot l_{\lfloor 6u+2+p \rfloor Q, \lfloor -p^2 \rfloor Q}(P)$	$(p^{12} - 1)/r$
BLS-12	$f_{u, Q}(P)$	$(p^{12} - 1)/r$

The naive calculation procedure of Optimal-Ate pairing is shown in Algorithm 1. In what follows, the calculation steps from 1 to 11, shown in Algorithm 1, is identified as Miller's Algorithm (MA) and step 12 is the final exponentiation (FE). Steps 2-7 are specially named as Miller's loop. Steps 3, 5, 7 are the line evaluation together with elliptic curve doubling (ECD) and addition (ECA) inside the Miller's loop and steps 9, 11 are the line evaluation outside the loop. These line evaluation steps are the key steps to accelerate the loop calculation. The authors extended the work of [20, 23] for KSS-16 curve to calculate *pseudo 8-sparse multiplication* described in Sect. 3. The ECA and ECD are also calculated efficiently in the twisted curve. The $Q_2 \leftarrow \lfloor p \rfloor Q$ term of step 8 is calculated by applying one skew Frobenius map over \mathbb{F}_{p^4} and $f_1 \leftarrow f^{p^3}$ of step 10 is calculated by applying one Frobenius map in $\mathbb{F}_{p^{16}}$. Step 12, FE is calculated by applying Ghammam et al.'s work for KSS-16 curve [16].

2.4 Twist of KSS-16 Curves

In the context of Type 3 pairing, there exists a *twisted curve* with a group of rational points of order r , isomorphic to the group where rational point $Q \in E(\mathbb{F}_{p^k})[r] \cap \text{Ker}(\pi_p - \lfloor p \rfloor)$ belongs to. This sub-field isomorphic rational point group includes a twisted isomorphic point of Q , typically denoted as $Q' \in E'(\mathbb{F}_{p^{k/d}})$, where k is the embedding degree and d is the twist degree.

Algorithm 1. Optimal Ate pairing on KSS-16 curve

Input: $u, P \in \mathbb{G}_1, Q \in \mathbb{G}'_2$
Output: (Q, P)

```

1  $f \leftarrow 1, T \leftarrow Q$ 
2 for  $i = \lfloor \log_2(u) \rfloor$  downto 1 do
3    $f \leftarrow f^2 \cdot l_{T,T}(P), T \leftarrow [2]T$ 
4   if  $u[i] = 1$  then
5      $f \leftarrow f \cdot l_{T,Q}(P), T \leftarrow T + Q$ 
6   if  $u[i] = -1$  then
7      $f \leftarrow f \cdot l_{T,-Q}(P), T \leftarrow T - Q$ 
8  $Q_1 \leftarrow [u]Q, Q_2 \leftarrow [p]Q$ 
9  $f \leftarrow f \cdot l_{Q_1, Q_2}(P)$ 
10  $f_1 \leftarrow f^{p^3}, f \leftarrow f \cdot f_1$ 
11  $f \leftarrow f \cdot l_{Q,Q}(P)$ 
12  $f \leftarrow f^{\frac{p^{16}-1}{r}}$ 
13 return  $f$ 

```

Since points on the twisted curve are defined over a smaller field than \mathbb{F}_{p^k} , therefore ECA and ECD become faster. However, when required in the Miller’s algorithm’s line evaluation, the points can be quickly mapped to points on $E(\mathbb{F}_{p^k})$. Since the pairing-friendly KSS-16 [19] curve has CM discriminant of $D = 1$ and $4|k$; therefore, quartic twist is available.

Quartic Twist. Let β be a certain quadratic non-residue in \mathbb{F}_{p^4} . The quartic twisted curve E' of KSS-16 curve E defined in Eq. (1) and their isomorphic mapping ψ_4 are given as follows:

$$\begin{aligned}
 E' &: y^2 = x^3 + ax\beta^{-1}, \quad a \in \mathbb{F}_p, \\
 \psi_4 &: E'(\mathbb{F}_{p^4})[r] \mapsto E(\mathbb{F}_{p^{16}})[r] \cap \text{Ker}(\pi_p - [p]), \\
 &(x, y) \mapsto (\beta^{1/2}x, \beta^{3/4}y), \tag{8}
 \end{aligned}$$

where $\text{Ker}(\cdot)$ denotes the kernel of the mapping and π_p denotes Frobenius mapping for rational point.

Table 4 shows the vector representation of $Q = (x_Q, y_Q) = (\beta^{1/2}x_{Q'}, \beta^{3/4}y_{Q'}) \in \mathbb{F}_{p^{16}}$ according to the given tower in Eq. (3). Here, $x_{Q'}$ and $y_{Q'}$ are the coordinates of rational point Q' on quartic twisted curve E' .

Table 4. Vector representation of $Q = (x_Q, y_Q) \in \mathbb{G}_2 \subset E(\mathbb{F}_{p^{16}})$

	1	α	β	$\alpha\beta$	γ	$\alpha\gamma$	$\beta\gamma$	$\alpha\beta\gamma$	ω	$\alpha\omega$	$\beta\omega$	$\alpha\beta\omega$	$\gamma\omega$	$\alpha\gamma\omega$	$\beta\gamma\omega$	$\alpha\beta\gamma\omega$
x_Q	0	0	0	0	b_4	b_5	b_6	b_7	0	0	0	0	0	0	0	0
y_Q	0	0	0	0	0	0	0	0	0	0	0	0	b_{12}	b_{13}	b_{14}	b_{15}

3 Proposal

3.1 Overview: Sparse and Pseudo-Sparse Multiplication

Aranha et al. [1, Sect. 4] and Costello et al. [10] have well optimized the Miller's algorithm in Jacobian coordinates by 6-sparse multiplication¹ for BN curve. Mori et al. [23] have shown the pseudo 8-sparse multiplication² for BN curve by adapting affine coordinates where the sextic twist is available. It is found that pseudo 8-sparse was efficient than 7-sparse and 6-sparse in Jacobian coordinates.

Let us consider $T = (\gamma x_{T'}, \gamma \omega y_{T'})$, $Q = (\gamma x_{Q'}, \gamma \omega y_{Q'})$ and $P = (x_P, y_P)$, where $x_P, y_P \in \mathbb{F}_p$ given in affine coordinates on the curve $E(\mathbb{F}_{p^{16}})$ such that $T' = (x_{T'}, y_{T'})$, $Q' = (x_{Q'}, y_{Q'})$ are in the twisted curve E' defined over \mathbb{F}_{p^4} . Let the elliptic curve doubling of $T + T = R(x_R, y_R)$. The 7-sparse multiplication for KSS-16 can be derived as follows.

$$\begin{aligned}
 l_{T,T}(P) &= (y_P - y_{T'}\gamma\omega) - \lambda_{T,T}(x_P - x_{T'}\gamma), \quad \text{when } T = Q, \\
 \lambda_{T,T} &= \frac{3x_{T'}^2\gamma^2 + a}{2y_{T'}\gamma\omega} = \frac{3x_{T'}^2\gamma\omega^{-1} + a(\gamma\omega)^{-1}}{2y_{T'}} = \frac{(3x_{T'}^2 + ac^{-1}\alpha\beta)\omega}{2y_{T'}} = \lambda'_{T,T}\omega, \\
 &\quad \text{since } \gamma\omega^{-1} = \omega, (\gamma\omega)^{-1} = \omega\beta^{-1}, \quad \text{and} \\
 a\beta^{-1} &= (a + 0\alpha + 0\beta + 0\alpha\beta)\beta^{-1} = a\beta^{-1} = ac^{-1}\alpha\beta, \quad \text{where } \alpha^2 = c.
 \end{aligned}$$

Now the line evaluation and ECD are obtained as follows:

$$\begin{aligned}
 l_{T,T}(P) &= y_P - x_P\lambda'_{T,T}\omega + (x_{T'}\lambda'_{T,T} - y_{T'})\gamma\omega, \\
 x_{2T'} &= (\lambda'_{T,T})^2\omega^2 - 2x_{T'}\gamma = ((\lambda'_{T,T})^2 - 2x_{T'})\gamma \\
 y_{2T'} &= (x_{T'}\gamma - x_{2T'}\gamma)\lambda'_{T,T}\omega - y_{T'}\gamma\omega = (x_{T'}\lambda'_{T,T} - x_{2T'}\lambda'_{T,T} - y_{T'})\gamma\omega.
 \end{aligned}$$

The above calculations can be optimized as follows:

$$\begin{aligned}
 A &= \frac{1}{2y_{T'}}, B = 3x_{T'}^2 + ac^{-1}, C = AB, D = 2x_{T'}, x_{2T'} = C^2 - D, \\
 E &= Cx_{T'} - y_{T'}, y_{2T'} = E - Cx_{2T'}, \\
 l_{T,T}(P) &= y_P + E\gamma\omega - Cx_P\omega = y_P + F\omega + E\gamma\omega,
 \end{aligned} \tag{9}$$

where $F = -Cx_P$.

The elliptic curve addition phase ($T \neq Q$) and line evaluation of $l_{T,Q}(P)$ can also be optimized similar to the above procedure. Let the elliptic curve addition of $T + Q = R(x_R, y_R)$.

¹ 6-Sparse refers the state when in a vector (multiplier/multiplicand), among the 12 coefficients 6 of them are zero..

² Pseudo 8-sparse refers to a certain length of vector's coefficients where instead of 8 zero coefficients, there are seven 0's and one 1 as coefficients..

$$\begin{aligned}
 l_{T,Q}(P) &= (y_p - y_{T'}\gamma\omega) - \lambda_{T,Q}(x_p - x_{T'}\gamma), \quad T \neq Q, \\
 \lambda_{T,Q} &= \frac{(y_{Q'} - y_{T'})\gamma\omega}{(x_{Q'} - x_{T'})\gamma} = \frac{(y_{Q'} - y_{T'})\omega}{x_{Q'} - x_{T'}} = \lambda'_{T,Q}\omega, \\
 x_R &= (\lambda'_{T,Q})^2\omega^2 - x_{T'}\gamma - x_{Q'}\gamma = ((\lambda'_{T,Q})^2 - x_{T'} - x_{Q'})\gamma \\
 y_R &= (x_{T'}\gamma - x_R\gamma)\lambda'_{T,Q}\omega - y_{T'}\gamma\omega = (x_{T'}\lambda'_{T,Q} - x_R\lambda'_{T,Q} - y_{T'})\gamma\omega.
 \end{aligned}$$

Representing the above line equations using variables as following:

$$\begin{aligned}
 A &= \frac{1}{x_{Q'} - x_{T'}}, B = y_{Q'} - y_{T'}, C = AB, D = x_{T'} + x_{Q'}, \\
 x_{R'} &= C^2 - D, E = Cx_{T'} - y_{T'}, y_{R'} = E - Cx_{R'}, \\
 l_{T,Q}(P) &= y_P + E\gamma\omega - Cx_P\omega = y_P + F\omega + E\gamma\omega, \\
 F &= -Cx_P,
 \end{aligned} \tag{10}$$

Here all the variables (A, B, C, D, E, F) are calculated as \mathbb{F}_{p^4} elements. The position of the y_P, E and F in $\mathbb{F}_{p^{16}}$ vector representation is defined by the basis element 1, γ and ω as shown in Table 4. Therefore, among the 16 coefficients of $l_{T,T}(P)$ and $l_{T,Q}(P) \in \mathbb{F}_{p^{16}}$, only 9 coefficients $y_P \in \mathbb{F}_p, Cx_P \in \mathbb{F}_{p^4}$ and $E \in \mathbb{F}_{p^4}$ are non-zero. The remaining 7 zero coefficients leads to an efficient multiplication, usually called sparse multiplication. This particular instance in KSS-16 curve is named as 7-sparse multiplication.

3.2 Pseudo 8-Sparse Multiplication for BN and BLS-12 Curve

Here we have followed Mori et al.'s [23] procedure to derive pseudo 8-sparse multiplication for the parameter settings of [3] for BN and BLS-12 curves. For the new parameter settings, the towering is given as Eq. (4) for both BN and BLS-12 curve. However, the curve form $E : y^2 = x^3 + b, b \in \mathbb{F}_p$ is identical for both BN and BLS-12 curve. The sextic twist obtained for these curves are also identical. Therefore, in what follows this paper will denote both of them as E_b defined over $\mathbb{F}_{p^{12}}$ (Table 5).

Sextic Twist of BN and BLS-12 Curve: Let $(\alpha + 1)$ be a certain quadratic and cubic non-residue in \mathbb{F}_{p^2} . The sextic twisted curve E'_b of curve E_b and their isomorphic mapping ψ_6 are given as follows:

$$\begin{aligned}
 E'_b &: y^2 = x^3 + b(\alpha + 1), \quad b \in \mathbb{F}_p, \\
 \psi_6 &: E'_b(\mathbb{F}_{p^2})[r] \mapsto E_b(\mathbb{F}_{p^{12}})[r] \cap \text{Ker}(\pi_p - [p]), \\
 (x, y) &\mapsto ((\alpha + 1)^{-1}x\beta, (\alpha + 1)^{-1}y\beta^2\gamma).
 \end{aligned} \tag{11}$$

The line evaluation and ECD/ECA can be obtained in affine coordinate for the rational point P and $Q', T' \in E'_b(\mathbb{F}_{p^2})$ as follows:

Elliptic curve addition when $T' \neq Q'$ and $T' + Q' = R'(x_{R'}, y_{R'})$

$$\begin{aligned}
 A &= \frac{1}{x_{Q'} - x_{T'}}, B = y_{Q'} - y_{T'}, C = AB, D = x_{T'} + x_{Q'}, \\
 x_{R'} &= C^2 - D, E = Cx_{T'} - y_{T'}, y_{R'} = E - Cx_{R'}, \\
 l_{T',Q'}(P) &= y_P + (\alpha + 1)^{-1}E\beta\gamma - (\alpha + 1)^{-1}Cx_P\beta^2\gamma,
 \end{aligned} \tag{12a}$$

$$y_P^{-1}l_{T',Q'}(P) = 1 + (\alpha + 1)^{-1}Ey_P^{-1}\beta\gamma - (\alpha + 1)^{-1}Cx_Py_P^{-1}\beta^2\gamma, \tag{12b}$$

Table 5. Vector representation of $Q = (x_Q, y_Q) \in \mathbb{G}_2 \subset E(\mathbb{F}_{p^{12}})$ vector representation

	1	α	β	$\alpha\beta$	β^2	$\alpha\beta^2$	γ	$\alpha\gamma$	$\beta\gamma$	$\alpha\beta\gamma$	$\beta^2\gamma$	$\alpha\beta^2\gamma$
x_Q	0	0	0	0	b_4	b_5	0	0	0	0	0	0
y_Q	0	0	0	0	0	0	0	0	b_8	b_9	0	0

Elliptic curve doubling when $T' = Q'$

$$A = \frac{1}{2y_{T'}}, B = 3x_{T'}^2, C = AB, D = 2x_{T'}, x_{2T'} = C^2 - D,$$

$$E = Cx_{T'} - y_{T'}, y_{2T'} = E - Cx_{2T'},$$

$$l_{T', T'}(P) = y_P + (\alpha + 1)^{-1}E\beta\gamma - (\alpha + 1)^{-1}Cx_P\beta^2\gamma, \quad (13a)$$

$$y_P^{-1}l_{T', T'}(P) = 1 + (\alpha + 1)^{-1}Ey_P^{-1}\beta\gamma - (\alpha + 1)^{-1}Cx_Py_P^{-1}\beta^2\gamma, \quad (13b)$$

The line evaluations of Eqs. (12b) and (13b) are identical and more sparse than Eqs. (12a) and (13a). Such sparse form comes with a cost of computation overhead. But such overhead can be minimized by the following isomorphic mapping, which also accelerates the Miller's loop iteration.

Isomorphic mapping of $P \in \mathbb{G}_1 \mapsto \hat{P} \in \mathbb{G}_1'$:

$$\begin{aligned} \hat{E} : y^2 &= x^3 + b\hat{z}, \\ \hat{E}(\mathbb{F}_p)[r] &\mapsto E(\mathbb{F}_p)[r], \\ (x, y) &\mapsto (\hat{z}^{-1}x, \hat{z}^{-3/2}y), \end{aligned} \quad (14)$$

where $\hat{z} \in \mathbb{F}_p$ is a quadratic and cubic residue in \mathbb{F}_p . Equation (14) maps rational point P to $\hat{P}(x_{\hat{P}}, y_{\hat{P}})$ such that $(x_{\hat{P}}, y_{\hat{P}}^{-1}) = 1$. The twist parameter \hat{z} is obtained as:

$$\hat{z} = (x_P y_P^{-1})^6. \quad (15)$$

From the Eq. (15) \hat{P} and \hat{Q}' is given as

$$\hat{P}(x_{\hat{P}}, y_{\hat{P}}) = (x_P z^{-1}, y_P z^{-3/2}) = (x_P^3 y_P^{-2}, x_P^3 y_P^{-2}), \quad (16a)$$

$$\hat{Q}'(x_{\hat{Q}'}, y_{\hat{Q}'}) = (x_P^2 y_P^{-2} x_{Q'}, x_P^3 y_P^{-3} y_{Q'}). \quad (16b)$$

Using Eqs. (16a) and (16b) the line evaluation of Eq. (13b) becomes

$$\begin{aligned} y_{\hat{P}}^{-1}l_{\hat{T}', \hat{T}'}(\hat{P}) &= 1 + (\alpha + 1)^{-1}Ey_{\hat{P}}^{-1}\beta\gamma - (\alpha + 1)^{-1}Cx_{\hat{P}}y_{\hat{P}}^{-1}\beta^2\gamma, \\ \hat{l}_{\hat{T}', \hat{T}'}(\hat{P}) &= 1 + (\alpha + 1)^{-1}Ey_{\hat{P}}^{-1}\beta\gamma - (\alpha + 1)^{-1}C\beta^2\gamma. \end{aligned} \quad (17a)$$

The Eq. (12b) becomes similar to Eq. (17a). The calculation overhead can be reduced by pre-computation of $(\alpha + 1)^{-1}$, $y_{\hat{P}}^{-1}$ and \hat{P} , \hat{Q}' mapping using x_P^{-1} and y_P^{-1} as shown by Mori et al. [23].

Finally, pseudo 8-sparse multiplication for BN and BLS-12 is given in

Algorithm 2. Pseudo 8-sparse multiplication for BN and BLS-12 curves**Input:** $a, b \in \mathbb{F}_{p^{12}}$

$$a = (a_0 + a_1\beta + a_2\beta^2) + (a_3 + a_4\beta + a_5\beta^2)\gamma, \quad b = 1 + b_4\beta\gamma + b_5\beta^2\gamma$$

where $a_i, b_j, c_i \in \mathbb{F}_{p^2} (i = 0, \dots, 5, j = 4, 5)$ **Output:** $c = ab = (c_0 + c_1\beta + c_2\beta^2) + (c_3 + c_4\beta + c_5\beta^2)\gamma \in \mathbb{F}_{p^{12}}$

$$1 \quad c_4 \leftarrow a_0 \times b_4, \quad t_1 \leftarrow a_1 \times b_5, \quad t_2 \leftarrow a_0 + a_1, \quad S_0 \leftarrow b_4 + b_5$$

$$2 \quad c_5 \leftarrow t_2 \times S_0 - (c_4 + t_1), \quad t_2 \leftarrow a_2 \times b_5, \quad t_2 \leftarrow t_2 \times (\alpha + 1)$$

$$3 \quad c_4 \leftarrow c_4 + t_2, \quad t_0 \leftarrow a_2 \times b_4, \quad t_0 \leftarrow t_0 + t_1$$

$$4 \quad c_3 \leftarrow t_0 \times (\alpha + 1), \quad t_0 \leftarrow a_3 \times b_4, \quad t_1 \leftarrow a_4 \times b_5, \quad t_2 \leftarrow a_3 + a_4$$

$$5 \quad t_2 \leftarrow t_2 \times S_0 - (t_0 + t_1)$$

$$6 \quad c_0 \leftarrow t_2 \times (\alpha + 1), \quad t_2 \leftarrow a_5 \times b_4, \quad t_2 \leftarrow t_1 + t_2$$

$$7 \quad c_1 \leftarrow t_2 \times (\alpha + 1), \quad t_1 \leftarrow a_5 \times b_5, \quad t_1 \leftarrow t_1 \times (\alpha + 1)$$

$$8 \quad c_2 \leftarrow t_0 + t_1$$

$$9 \quad c \leftarrow c + a$$

$$10 \quad \text{return } c = (c_0 + c_1\beta + c_2\beta^2) + (c_3 + c_4\beta + c_5\beta^2)\gamma$$

3.3 Pseudo 8-Sparse Multiplication for KSS-16 Curve

The main idea of *pseudo 8-sparse multiplication* is finding more sparse form of Eqs. (9) and (10), which allows to reduce the number of multiplication of $\mathbb{F}_{p^{16}}$ vector during Miller's algorithm evaluation. To obtains the same, y_P^{-1} is multiplied to both side of Eqs. (9) and (10), since y_P remains the same through the Miller's algorithms loop calculation.

$$y_P^{-1}l_{T,T}(P) = 1 - Cx_Py_P^{-1}\omega + Ey_P^{-1}\gamma\omega, \quad (18a)$$

$$y_P^{-1}l_{T,Q}(P) = 1 - Cx_Py_P^{-1}\omega + Ey_P^{-1}\gamma\omega, \quad (18b)$$

Although the Eqs. (18a) and (18b) do not get more sparse, but 1st coefficient becomes 1. Such vector is titled as *pseudo sparse form* in this paper. This form realizes more efficient $\mathbb{F}_{p^{16}}$ vectors multiplication in Miller's loop. However, the Eq. (18b) creates more computation overhead than Eq. (10), i.e., computing $y_P^{-1}l_{T,Q}(P)$ in the left side and $x_Py_P^{-1}$, Ey_P^{-1} on the right. The same goes between Eqs. (9) and (18a). Since the computation of Eqs. (18a) and (18b) are almost identical, therefore the rest of the paper shows the optimization technique for Eq. (18a). To overcome these overhead computations, the following techniques can be applied.

- $x_Py_P^{-1}$ is omitted by applying further isomorphic mapping of $P \in \mathbb{G}_1$.
- y_P^{-1} can be pre-computed. Therefore, the overhead calculation of Ey_P^{-1} will cost only $2 \mathbb{F}_p$ multiplication.
- $y_P^{-1}l_{T,T}(P)$ doesn't effect the pairing calculation cost since the final exponentiation cancels this multiplication by $y_P^{-1} \in \mathbb{F}_p$.

To overcome the $Cx_Py_P^{-1}$ calculation cost, $x_Py_P^{-1} = 1$ is expected. To obtain $x_Py_P^{-1} = 1$, the following isomorphic mapping of $P = (x_P, y_P) \in \mathbb{G}_1$ is introduced.

Isomorphic Map of $P = (x_P, y_P) \rightarrow \bar{P} = (x_{\bar{P}}, y_{\bar{P}})$. Although the KSS-16 curve is typically defined over $\mathbb{F}_{p^{16}}$ as $E(\mathbb{F}_{p^{16}})$, but for efficient implementation of Optimal-Ate pairing, certain operations are carried out in a quartic twisted isomorphic curve E' defined over \mathbb{F}_{p^4} as shown in Sect. 2.4. For the same, let us consider $\bar{E}(\mathbb{F}_{p^4})$ is isomorphic to $E(\mathbb{F}_{p^4})$ and certain $z \in \mathbb{F}_p$ as a quadratic residue (QR) in \mathbb{F}_{p^4} . A generalized mapping between $E(\mathbb{F}_{p^4})$ and $\bar{E}(\mathbb{F}_{p^4})$ can be given as follows:

$$\begin{aligned} \bar{E} : y^2 &= x^3 + az^{-2}x, \\ \bar{E}(\mathbb{F}_{p^4})[r] &\mapsto E(\mathbb{F}_{p^4})[r], \\ (x, y) &\mapsto (z^{-1}x, z^{-3/2}y), \\ \text{where } z, z^{-1}, z^{-3/2} &\in \mathbb{F}_p. \end{aligned} \quad (19)$$

The mapping considers $z \in \mathbb{F}_p$ is a quadratic residue over \mathbb{F}_{p^4} which can be shown by the fact that $z^{(p^4-1)/2} = 1$ as follows:

$$\begin{aligned} z^{(p^4-1)/2} &= z^{(p-1)(p^3+p^2+p+1)/2} \\ &= 1^{(p^3+p^2+p+1)/2} \\ &= 1 \quad \text{QR} \in \mathbb{F}_{p^4}. \end{aligned} \quad (20)$$

Therefore, z is a quadratic residue over \mathbb{F}_{p^4} .

Now based on $P = (x_P, y_P)$ be the rational point on curve E , the considered isomorphic mapping of Eq. (19) can find a certain isomorphic rational point $\bar{P} = (x_{\bar{P}}, y_{\bar{P}})$ on curve \bar{E} as follows:

$$\begin{aligned} y_{\bar{P}}^2 &= x_{\bar{P}}^3 + ax_{\bar{P}}, \\ y_{\bar{P}}^2 z^{-3} &= x_{\bar{P}}^3 z^{-3} + ax_{\bar{P}} z^{-3}, \\ (y_{\bar{P}} z^{-3/2})^2 &= (x_{\bar{P}} z^{-1})^3 + az^{-2} x_{\bar{P}} z^{-1}, \end{aligned} \quad (21)$$

where $\bar{P} = (x_{\bar{P}}, y_{\bar{P}}) = (x_P z^{-1}, y_P z^{-3/2})$ and the general form of the curve \bar{E} is given as follows:

$$y^2 = x^3 + az^{-2}x. \quad (22)$$

To obtain the target relation $x_{\bar{P}} y_{\bar{P}}^{-1} = 1$ from above isomorphic map and rational point \bar{P} , let us find isomorphic twist parameter z as follows:

$$\begin{aligned} x_{\bar{P}} y_{\bar{P}}^{-1} &= 1 \\ z^{-1} x_P (z^{-3/2} y_P)^{-1} &= 1 \\ z^{1/2} (x_P \cdot y_P^{-1}) &= 1 \\ z &= (x_P^{-1} y_P)^2. \end{aligned} \quad (23)$$

Now using $z = (x_P^{-1} y_P)^2$ and Eq. (21), \bar{P} can be obtained as

$$\bar{P}(x_{\bar{P}}, y_{\bar{P}}) = (x_P z^{-1}, y_P z^{-3/2}) = (x_P^3 y_P^{-2}, x_P^3 y_P^{-2}), \quad (24)$$

where the x and y coordinates of \bar{P} are equal. For the same isomorphic map we can obtain \bar{Q} on curve \bar{E} defined over $\mathbb{F}_{p^{12}}$ as follows:

$$\bar{Q}(x_{\bar{Q}}, y_{\bar{Q}}) = (z^{-1}x_{Q'}\gamma, z^{-3/2}y_{Q'}\gamma\omega), \tag{25}$$

where from Eq. (8), $Q'(x_{Q'}, y_{Q'})$ is obtained in quartic twisted curve E' .

At this point, to use \bar{Q} with \bar{P} in line evaluation we need to find another isomorphic map that will map $\bar{Q} \mapsto \bar{Q}'$, where \bar{Q}' is the rational point on curve \bar{E}' defined over \mathbb{F}_{p^4} . Such \bar{Q}' and \bar{E}' can be obtained from \bar{Q} of Eq. (25) and curve \bar{E} from Eq. (22) as follows:

$$\begin{aligned} (z^{-3/2}y_{Q'}\gamma\omega)^2 &= (z^{-1}x_{Q'}\gamma)^3 + az^{-2}z^{-1}x_{Q'}\gamma, \\ (z^{-3/2}y_{Q'})^2\gamma^2\omega^2 &= (z^{-1}x_{Q'})^3\gamma^3 + az^{-2}z^{-1}x_{Q'}\gamma, \\ (z^{-3/2}y_{Q'})^2\beta\gamma &= (z^{-1}x_{Q'})^3\beta\gamma + az^{-2}z^{-1}x_{Q'}\gamma, \\ (z^{-3/2}y_{Q'})^2 &= (z^{-1}x_{Q'})^3 + az^{-2}\beta^{-1}z^{-1}x_{Q'}. \end{aligned}$$

From the above equations, \bar{E}' and \bar{Q}' are given as,

$$\bar{E}' : y_{\bar{Q}'}^2 = x_{\bar{Q}'}^3 + a(z^2\beta)^{-1}x_{\bar{Q}'}. \tag{26}$$

$$\begin{aligned} \bar{Q}'(x_{\bar{Q}'}, y_{\bar{Q}'}) &= (z^{-1}x_{Q'}, z^{-3/2}y_{Q'}), \\ &= (x_{Q'}x_P^2y_P^{-2}, y_{Q'}x_P^3y_P^{-3}). \end{aligned} \tag{27}$$

Now, applying \bar{P} and \bar{Q}' , the line evaluation of Eq. (18b) becomes as follows:

$$\begin{aligned} y_{\bar{P}}^{-1}l_{\bar{P}', \bar{Q}'}(\bar{P}) &= 1 - C(x_{\bar{P}}y_{\bar{P}}^{-1})\gamma + Ey_{\bar{P}}^{-1}\gamma\omega, \\ \bar{l}_{\bar{P}', \bar{Q}'}(\bar{P}) &= 1 - C\gamma + E(x_{\bar{P}}^{-3}y_{\bar{P}}^2)\gamma\omega, \end{aligned} \tag{28}$$

where $x_{\bar{P}}y_{\bar{P}}^{-1} = 1$ and $y_{\bar{P}}^{-1} = z^{3/2}y_P^{-1} = (x_P^{-3}y_P^2)$. The Eq. (18a) becomes the same as Eq. (28). Compared to Eq. (18b), the Eq. (28) will be faster while using in Miller’s loop in combination of the pseudo 8-sparse multiplication shown in Algorithm 2. However, to get the above form, we need the following pre-computations once in every Miller’s Algorithm execution.

- Computing \bar{P} and \bar{Q}' ,
- $(x_P^{-3}y_P^2)$ and
- z^{-2} term from curve \bar{E}' of Eq. (26).

The above terms can be computed from x_P^{-1} and y_P^{-1} by utilizing Montgomery trick [22], as shown in Algorithm 3. The pre-computation requires 21 multiplication, 2 squaring and 1 inversion in \mathbb{F}_p and 2 multiplication, 3 squaring in \mathbb{F}_{p^4} .

The overall mapping and the curve obtained in the twisting process is shown in the Fig. 1.

Finally the Algorithm 4 shows the derived pseudo 8-sparse multiplication.

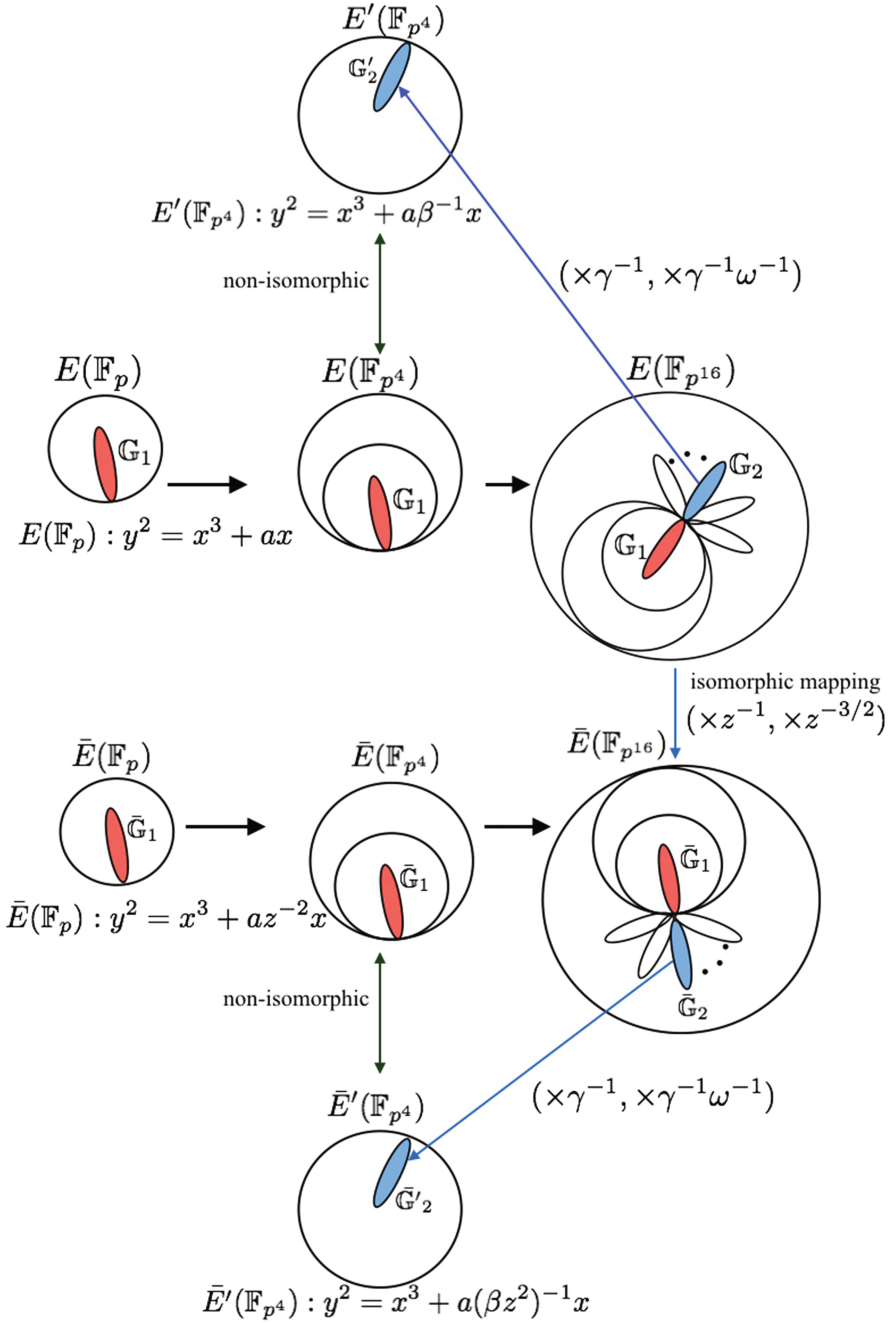


Fig. 1. Overview of the twisting process to get pseudo sparse form in KSS-16 curve.

Algorithm 3. Pre-calculation and mapping $P \mapsto \bar{P}$ and $Q' \mapsto \bar{Q}'$

Input: $P = (x_P, y_P) \in \mathbb{G}_1, Q' = (x_{Q'}, y_{Q'}) \in \mathbb{G}'_2$ **Output:** $\bar{Q}', \bar{P}, y_{\bar{P}}^{-1}, (z)^{-2}$

```

1  $A \leftarrow (x_P y_P)^{-1}$ 
2  $B \leftarrow Ax_P^2$ 
3  $C \leftarrow Ay_P$ 
4  $D \leftarrow B^2$ 
5  $x_{\bar{Q}'} \leftarrow Dx_{Q'}$ 
6  $y_{\bar{Q}'} \leftarrow BDy_{Q'}$ 
7  $x_{\bar{P}}, y_{\bar{P}} \leftarrow Dx_P$ 
8  $y_{\bar{P}}^{-1} \leftarrow C^3 y_P^2$ 
9  $z^{-2} \leftarrow D^2$ 
10 return  $\bar{Q}' = (x_{\bar{Q}'}, y_{\bar{Q}'}), \bar{P} = (x_{\bar{P}}, y_{\bar{P}}), y_{\bar{P}}^{-1}, z^{-2}$ 

```

Algorithm 4. Pseudo 8-sparse multiplication for KSS-16 curve

Input: $a, b \in \mathbb{F}_{p^{16}}$ $a = (a_0 + a_1\gamma) + (a_2 + a_3\gamma)\omega, b = 1 + (b_2 + b_3\gamma)\omega$ $a = (a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3), b = 1 + b_2\omega + b_3\omega^3$ **Output:** $c = ab = (c_0 + c_1\gamma) + (c_3 + c_4\gamma)\omega \in \mathbb{F}_{p^{16}}$

```

1  $t_0 \leftarrow a_3 \times b_3 \times \beta, t_1 \leftarrow a_2 \times b_2, t_4 \leftarrow b_2 + b_3, c_0 \leftarrow (a_2 + a_3) \times t_4 - t_1 - t_0$ 
2  $c_1 \leftarrow t_1 + t_0 \times \beta$ 
3  $t_2 \leftarrow a_1 \times b_3, t_3 \leftarrow a_0 \times b_2, c_2 \leftarrow t_3 + t_2 \times \beta$ 
4  $t_4 \leftarrow (b_2 + b_3), c_3 \leftarrow (a_0 + a_1) \times t_4 - t_3 - t_2$ 
5  $c \leftarrow c + a$ 
6 return  $c = (c_0 + c_1\gamma) + (c_3 + c_4\gamma)\omega$ 

```

3.4 Final Exponentiation

Scott et al. [27] show the process of efficient final exponentiation (FE) $f^{p^k-1/r}$ by decomposing the exponent using cyclotomic polynomial Φ_k as

$$(p^k - 1)/r = (p^{k/2} - 1) \cdot (p^{k/2} + 1) / \Phi_k(p) \cdot \Phi_k(p) / r. \quad (29)$$

The 1st two terms of the right part are denoted as easy part since it can be easily calculated by Frobenius mapping and one inversion in affine coordinates. The last term is called hard part which mostly affects the computation performance. According to Eq. (29), the exponent decomposition of the target curves is shown in Table 6.

This paper carefully concentrates on Miller's algorithm for comparison and making pairing efficient. However, to verify the correctness of the bilinearity property, the authors made a "not state-of-art" implementation of Fuentes et al.'s work [14] for BN curve case and Ghammam's et al.'s works [16, 17] for KSS-16 and BLS-12 curves. For scalar multiplication by prime p , i.e., $p[Q]$ or $[p^2]Q$, skew Frobenius map technique by Sakemi et al. [26] is adapted.

Table 6. Exponents of final exponentiation in pairing

Curve	Final exponent	Easy part	Hard part
KSS-16	$\frac{p^{16}-1}{r}$	$p^8 - 1$	$\frac{p^8+1}{r}$
BN, BLS-12	$\frac{p^{12}-1}{r}$	$(p^6 - 1)(p^2 + 1)$	$\frac{p^4 - p^2 + 1}{r}$

4 Experimental Result Evaluation

This section gives details of the experimental implementation. The source code can be found in Github³. The code is not an optimal code, and the sole purpose of it compare the Miller’s algorithm among the curve families and validate the estimation of [3]. Table 7 shows implementation environment. Parameters chosen from [3] is shown in Table 8. Table 9 shows execution time for Miller’s algorithm implementation in millisecond for a single Optimal-Ate pairing. Results here are the average of 10 pairing operation. From the result, we find that Miller’s algorithm took the least time for KSS-16. And the time is almost closer to BLS-12. The Miller’s algorithm is about 1.7 times faster in KSS-16 than BN curve. Table 12 shows that the complexity of this implementation concerning the number of \mathbb{F}_p multiplication and squaring and the estimation of [3] are almost coherent for Miller’s algorithm. Table 12 also show that our derived pseudo 8-sparse multiplication for KSS-16 takes fewer \mathbb{F}_p multiplication than Zhang et al.’s estimation [31]. The execution time of Miller’s algorithm also goes with this estimation [3], that means KSS-16 and BLS-12 are more efficient than BN curve. Table 10 shows the complexity of Miller’s algorithm for the target curves in \mathbb{F}_p operations count.

Table 7. Computational Environment

CPU ^a	Memory	Compiler	OS	Language	Library
Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz	4 GB	GCC 5.4.0	Ubuntu 16.04 LTS	C	GMP v 6.1.0 [18]

^aOnly single core is used from two cores

The operation counted in Table 10 are based on the counter in implementation code. For the implementation of big integer arithmetic `mpz_t` data type of GMP [18] library has been used. For example, multiplication between 2 `mpz_t` variables are counted as \mathbb{F}_p multiplication and multiplication between one `mpz_t` and one “unsigned long” integer can also be treated as \mathbb{F}_p multiplication. Basis multiplication refers to the vector multiplication such as $(a_o + a_1\alpha)\alpha$ where $a_0, a_1 \in \mathbb{F}_p$ and α is the basis element in \mathbb{F}_{p^2} .

³ <https://github.com/eNipu/pairingma128.git>.

Table 8. Selected parameters for 128-bit security level [3]

Curve	u	HW(u)	$\lfloor \log_2 u \rfloor$	$\lfloor \log_2 p(u) \rfloor$	$\lfloor \log_2 r(u) \rfloor$	$\lfloor \log_2 p^k \rfloor$
KSS-16	$u = 2^{35} - 2^{32} - 2^{18} + 2^8 + 1$	5	35	339	263	5424
BN	$u = 2^{114} + 2^{101} - 2^{14} - 1$	4	115	462	462	5535
BLS-12	$u = -2^{77} + 2^{50} + 2^{33}$	3	77	461	308	5532

Table 9. Comparative results of Miller’s Algorithm in [ms].

	KSS-16	BN	BLS-12
Miller’s Algorithm	4.41	7.53	4.91

Table 10. Complexity of this implementation in \mathbb{F}_p for Miller’s algorithm [single pairing operation]

	Multiplication		Squaring	Addition/Subtraction	Basis multiplication	Inversion
	mpz_t * mpz_t	mpz_t * ui				
KSS-16	6162	144	903	23956	3174	43
BN	10725	232	157	35424	3132	125
BLS-12	6935	154	113	23062	2030	80

Table 11. Final exponentiation time (not state-of-art) in [ms]

	KSS-16	BN	BLS-12
Final exponentiation	17.32	11.65	12.03

Table 12. Complexity comparison of Miller’s algorithm between this implementation and Barbulescu et al.’s [3] estimation [Multiplication + Squaring in \mathbb{F}_p]

	KSS-16	BN	BLS-12
Barbulescu et al. [3]	$7534M_p$	$12068M_p$	$7708M_p$
This implementation	$7209M_p$	$11114M_p$	$7202M_p$

As said before, this work is focused on Miller’s algorithm. However, the authors made a “not state-of-art” implementation of some final exponentiation algorithms [14, 16, 17]. Table 11 shows the total final exponentiation time in [ms]. Here final exponentiation of KSS-16 is slower than BN and BLS-12. We have applied square and multiply technique for exponentiation by integer u in the hard part since the integer u given in the sparse form. However, Barbulescu et al. [3] mentioned that availability of compressed squaring [1] for KSS-16 will lead a fair comparison using final exponentiation.

5 Conclusion and Future Work

This paper has presented two major ideas.

- Finding efficient Miller’s algorithm implementation technique for Optimal-Ate pairing for the less studied KSS-16 curve. The author’s presented pseudo 8-sparse multiplication technique for KSS-16. They also extended such multiplication for BN and BLS-12 according to [23] for the new parameter.
- Verifying Barbulescu and Duquesne’s conclusion [3] for calculating Optimal-Ate pairing at 128-bit security level; that is, BLS-12 and less studied KSS-16 curves are more efficient choices than well studied BN curves for new parameters. This paper finds that Barbulescu and Duquesne’s conclusion on BLS-12 is correct as it takes the less time for Miller’s algorithm. Applying the derived pseudo 8-sparse multiplication, Miller’s algorithm in KSS-16 is also more efficient than BN.

As a prospective work authors would like to evaluate the performance by finding compressed squaring for KSS-16’s final exponentiation along with scalar multiplication of \mathbb{G}_1 , \mathbb{G}_2 and exponentiation of \mathbb{G}_T . The execution time for the target environment can be improved by a careful implementation using assembly language for prime field arithmetic.

Acknowledgments. This work was partially supported by the Strategic Information and Communications R&D Promotion Programme (SCOPE) of Ministry of Internal Affairs and Communications, Japan and The French projects ANR-16-CE39-0012 “SafeTLS” and ANR-11-LABX-0020-01 “Centre Henri Lebesgue”.

References

1. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster explicit formulas for computing pairings over ordinary curves. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 48–68. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_5
2. Bailey, D.V., Paar, C.: Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *J. Cryptol.* **14**(3), 153–176 (2001)
3. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. *Cryptology ePrint Archive*, Report 2017/334 (2017). <http://eprint.iacr.org/2017/334>
4. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 257–267. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36413-7_19
5. 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
6. Benger, N., Scott, M.: Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In: Hasan, M.A., Hellesteth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 180–195. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13797-6_13

7. 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
8. 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
9. Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press, Boca Raton (2005)
10. Costello, C., Lange, T., Naehrig, M.: Faster pairing computations on curves with high-degree twists. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 224–242. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_14
11. Devegili, A.J., O’Higeartaigh, C., Scott, M., Dahab, R.: Multiplication and squaring on pairing-friendly fields. IACR Cryptology ePrint Archive 2006, 471 (2006)
12. Duquesne, S., Mrabet, N.E., Haloui, S., Rondepierre, F.: Choosing and generating parameters for low level pairing implementation on BN curves. Cryptology ePrint Archive, Report 2015/1212 (2015). <http://eprint.iacr.org/2015/1212>
13. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. J. Cryptol. **23**(2), 224–280 (2010)
14. Fuentes-Castañeda, L., Knapp, E., Rodríguez-Henríquez, F.: Faster hashing to \mathbb{G}_2 . In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 412–430. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_25
15. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Appl. Math. **156**(16), 3113–3121 (2008)
16. Ghammam, L., Fouotsa, E.: Adequate elliptic curves for computing the product of n pairings. In: Duquesne, S., Petkova-Nikova, S. (eds.) WAIFI 2016. LNCS, vol. 10064, pp. 36–53. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-55227-9_3
17. Ghammam, L., Fouotsa, E.: On the computation of the optimal ate pairing at the 192-bit security level. Cryptology ePrint Archive, Report 2016/130 (2016). <http://eprint.iacr.org/2016/130>
18. Granlund, T., the GMP development team: GNU MP: the GNU Multiple Precision Arithmetic Library, 6.1.0 edn. (2015). <http://gmplib.org>
19. Kachisa, E.J., Schaefer, E.F., Scott, M.: Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209, pp. 126–135. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85538-5_9
20. Khandaker, M.A.-A., Ono, H., Nogami, Y., Shirase, M., Duquesne, S.: An improvement of optimal ate pairing on KSS curve with pseudo 12-sparse multiplication. In: Hong, S., Park, J.H. (eds.) ICISC 2016. LNCS, vol. 10157, pp. 208–219. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53177-9_11
21. Kim, T., Barbulescu, R.: Extended tower number field sieve: a new complexity for the medium prime case. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 543–571. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_20
22. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Math. Comput. **48**(177), 243–264 (1987)

23. Mori, Y., Akagi, S., Nogami, Y., Shirase, M.: Pseudo 8-sparse multiplication for efficient ate-based pairing on Barreto-Naehrig curve. In: Cao, Z., Zhang, F. (eds.) Pairing 2013. LNCS, vol. 8365, pp. 186–198. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04873-4_11
24. Okamoto, T., Takashima, K.: Fully secure functional encryption with general relations from the decisional linear assumption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 191–208. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_11
25. Sakai, R.: Cryptosystems based on pairing. In: The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, pp. 26–28, January 2000
26. Sakemi, Y., Nogami, Y., Okeya, K., Kato, H., Morikawa, Y.: Skew frobenius map and efficient scalar multiplication for pairing-based cryptography. In: Franklin, M.K., Hui, L.C.K., Wong, D.S. (eds.) CANS 2008. LNCS, vol. 5339, pp. 226–239. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89641-8_16
27. Scott, M., Benger, N., Charlemagne, M., Dominguez Perez, L.J., Kachisa, E.J.: On the final exponentiation for calculating pairings on ordinary elliptic curves. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 78–88. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03298-1_6
28. Silverman, J.H., Cornell, G., Artin, M.: Arithmetic Geometry. Springer, Heidelberg (1986)
29. Vercauteren, F.: Optimal pairings. IEEE Trans. Inf. Theory **56**(1), 455–461 (2010)
30. Weil, A., et al.: Numbers of solutions of equations in finite fields. Bull. Amer. Math. Soc. **55**(5), 497–508 (1949)
31. Zhang, X., Lin, D.: Analysis of optimum pairing products at high security levels. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 412–430. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_24

Fast Scalar Multiplication for Elliptic Curves over Binary Fields by Efficiently Computable Formulas

Saud Al Musa and Guangwu Xu^(✉)

Department of EE and CS, University of Wisconsin-Milwaukee, Milwaukee, USA
{salmusa, gxu4uwm}@uwm.edu

Abstract. This paper considers efficient scalar multiplication of elliptic curves over binary fields with a twofold purpose. Firstly, we derive the most efficient $3P$ formula in λ -projective coordinates and $5P$ formula in both affine and λ -projective coordinates. Secondly, extensive experiments have been conducted to test various multi-base scalar multiplication methods (e.g., greedy, ternary/binary, multi-base NAF, and tree-based) by integrating our fast formulas. The experiments show that our $3P$ and $5P$ formulas had an important role in speeding up the greedy, the ternary/binary, the multi-base NAF, and the tree-based methods over the NAF method. We also establish an efficient $3P$ formula for Koblitz curves and use it to construct an improved set for the optimal pre-computation of window TNAF.

Keywords: Binary elliptic curves · Point multiplication
Lambda coordinates · Efficient formulas · DBNS · MBNS

1 Introduction

Koblitz and Miller first introduced the use of an elliptic curve in public key cryptography [19, 26]. An elliptic curve cryptosystem is attractive for use because it has a shorter key length, and it is as secure as the larger key length in other public key cryptosystems. For instance, the shorter key length 283 bits in an elliptic curve cryptosystem is regarded as secure as the larger key length 3072 bits in an RSA cryptosystem [21].

The dominant operation in elliptic curve cryptography (ECC) cryptographic schemes is the scalar multiplication, which is an operation that adds a point to itself a large number of times. The research to increase the speed of this operation has attracted considerable attention ever since the discovery of the ECC. Many proposed methods have improved general exponentiation algorithms. The idea of presenting a scalar in a non-adjacent form (NAF) with signed coefficients has been a basic method to use. The sparse property of NAF participates in

Research supported in part by the National 973 Project of China (No. 2013CB834205).

minimizing the total number of point addition operations. Furthermore, the NAF method can be used with window width. The binary width- w NAF allows coefficients to be in $\{0, \pm 1, \pm 3, \dots, \pm 2^{w-1} - 1\}$, which makes the representation sparser. See [6, 17] for more details of the NAF and the window NAF methods.

Another faster method for scalar multiplication is to express the scalar in the double-base number system (DBNS). The DBNS with ternary and binary bases for scalar n is represented such that $n = \sum s 2^a 3^b$ where $a, b \geq 0, s \in \{-1, +1\}$. A natural extension of DBNS is multi-base number system (MBNS). The main advantage of using MBNS is that the scalar has a shorter average expansion length than its single-base average expansion length. As a result, the total number of point additions are minimized and that leads to faster scalar multiplication. A greater computational speed can be achieved if an efficient formula is available for scalar multiplication by an integer in the base. There are several methods for representing an integer in MBNS, including greedy method first proposed by Dimitrov et al. [8], ternary/binary method developed by Ciet et al. [7], tree-based method given by Doche and Habsieger [10], and multi-base NAF introduced by Longa in [22]. A scalar multiplication using MBNS expansion has been further researched in [2, 3, 9, 11, 23, 25, 27, 33].

In [18], Koblitz proposed a class of binary curves, what are now called Koblitz curves, for cryptographic use. Also Koblitz in [18] initiated a study of NAF of some algebraic integers using the Frobenius map τ . A very important extension to Koblitz's result was the window TNAF by Solinas [29] which reduces the computation for scalar multiplication dramatically. More computational properties of the window TNAF have been revealed by Blake et al. [4, 5]. Recently, Trost and Xu formulated and constructed an optimal pre-computation for window TNAF [30]. Some new formulas have been derived in [30] that require a fewer number of field operations.

Inversion operation is a very expensive operation in finite fields. The inversion to multiplication (\mathbf{I}/\mathbf{M}) ratio over binary fields is not fixed, and it gets affected by the inversion algorithm used and also the computing platform [16]. We usually assume the low \mathbf{I}/\mathbf{M} ratio is 5, and the high \mathbf{I}/\mathbf{M} ratio is 8 as suggested in [17]. Much effort on elliptic curve arithmetic has been made in working on different coordinate systems to avoid field inversion and achieve efficiency. A common way of avoiding expensive division is to change to projective coordinate systems. Besides standard projective coordinates, Jacobian and Chudnosky projective coordinate systems are also used for general curves. For curves over binary fields, López-Dahab (LD) coordinates [24] is a very efficient alternative. Recently, Oliveira, López, Aranha, and Rodríguez-Henríquez proposed a more efficient coordinate system for binary elliptic curves—the lambda representation (λ -coordinates) [28].

Devising efficient elliptic curve operations with small scalars have been of significant interest, e.g., reducing number of field operations for computing $3P$ and $5P$ for an elliptic curve point P . These operations are key to the fast computation by using DBNS and multi-base number representation. In [7], Ciet, Joye, Lauter, and Montgomery presented an efficient formula for $3P$ for both prime

curves and binary curves, and it takes 1 field inversion (**I**), 4 squarings (**S**), and 7 multiplications (**M**): $1\mathbf{I}+7\mathbf{M}+4\mathbf{S}$. A ternary/binary algorithm was also designed in [7] that utilizes $3P$ with an improved speed over the NAF method. For curves over a binary field, Dimitrov, Imbert, and Mishra gave an improved $3P$ formula that requires $1\mathbf{I}+6\mathbf{M}+3\mathbf{S}$ [8]. The most efficient formulas for computing $3P$ were given by Yu et al. in [32], their formula for binary field only needs $1\mathbf{I}+5\mathbf{M}+2\mathbf{S}$. In [27], Mishra and Dimitrov proposed the Multi-Base Number Representation for scalar multiplication. They derived an efficient $5P$ formula for binary curves with a small number of operations: $1\mathbf{I}+13\mathbf{M}+5\mathbf{S}$. Some concise formulas for Koblitz curves can be found in [30], e.g., for $(1-\tau)P$ and $(1+\tau)P$.

1.1 Our Contribution

In this paper, we consider the problem of fast scalar multiplication for binary elliptic curves. The main contribution of the paper is twofold. In the first part, we derive $3P$ and $5P$ efficient formulas for binary elliptic curves. In affine coordinates, our improved $5P$ formula uses $1\mathbf{I}+11\mathbf{M}+6\mathbf{S}$. Under the very promising λ -projective coordinate systems, we are able to set up efficient computation for $3P$ and $5P$ and their formulas cost $8\mathbf{M}+1\mathbf{M}_a+5\mathbf{S}$ and $13\mathbf{M}+1\mathbf{M}_a+8\mathbf{S}$ respectively, here \mathbf{M}_a denotes the cost of multiplication of a general field element with a fixed field element a (which is usually a coefficient of elliptic curve and has a small size). The derivation techniques for our $3P$ and $5P$ efficient formulas in λ -projective coordinates are not based on the $3P$ and $5P$ efficient formulas in affine coordinates. λ -coordinates system has its own affine coordinates, which is called λ -affine coordinates. Thus, it is necessary to find first a formula in λ -affine coordinates that leads to an efficient formula in λ -projective coordinates. The derived $3P$ and $5P$ efficient formulas in this paper are state of the art, and to the best of our knowledge, we are the first to present them in λ -projective coordinates. More precisely, our $3P$ λ -projective coordinates greatly improves that using LD projective coordinates [31] and Jacobian projective coordinates [8]. A projective coordinate formula for $5P$ seems not available in literature.

The second part of our contribution is conducting extensive performance comparison tests for the MBNS methods in λ -coordinates. The MBNS methods are one of the best applications that shows the importance of our $3P$ and $5P$ formulas in speeding up scalar multiplication operations. The investigated MBNS methods are the greedy, the ternary/binary, the multi-base NAF, and the tree-based [7, 8, 10, 23]. Our tests compare these methods using our efficient formulas with respect to three characteristics: the expansion length, the total number of multiplications, and the running time. Other comparison tests in [7, 8, 10, 23] emphasize the expansion length and the total number of multiplications. We include the running time test since it takes into account the time of converting integer n to a multi-base chain. To the best of our knowledge, we are the first study that compares the performance of the MBNS methods with the NAF method in λ -coordinates.

Our comparison test in terms of the total number of multiplications shows the greedy, the ternary/binary, the multi-base NAF, and the tree-based methods

speed up to 10%, 8%, 12%, and 15% over the NAF method. Our running time test shows they speed up to 7%, 9%, 12%, and 15% over the NAF method. The running time test of the greedy method gives less percentage of improvement than the comparison test in terms of the total number of multiplications. The reason for that is the running time test considers the time of converting integer n to a multi-base chain, which implies the greedy method has a higher conversion cost than other MBNS methods.

Some of the ideas for computing $3P$ also lead an improvement of the optimal pre-computation of window TNAF for Koblitz curves [30]. By efficient formulas for the pre-computed points in the forms of $P - \tau(P)$, $P + \tau(P)$, $P - \tau^2(P)$ and $3P$ and working with the λ -projective coordinates, we show the performance of the optimal pre-computation of window TNAF with these efficient formulas gets 48%, 24% and 11% faster for window width 4, 5 and 6 respectively.

The rest of the paper is organized into five sections. Efficient formulas for $3P$ and $5P$ are given in Sect. 2. In Sect. 3, we briefly review several existing MBNS methods, and we conduct comparison tests for these methods using our efficient formulas. In Sect. 4, we briefly review the optimal pre-computation of window TNAF for Koblitz curves, and we propose $3P$ efficient formula to the improved set of the optimal pre-computation of window TNAF with experiments. Finally, the paper is concluded in Sect. 5.

2 Formulas for $3P$ and $5P$ on Binary Elliptic Curves

A non-supersingular elliptic curve E over a binary field \mathbb{F}_{2^m} can be represented by the simplified Weierstrass equation

$$E : y^2 + xy = x^3 + ax^2 + b, \quad (1)$$

where $a, b \in \mathbb{F}_{2^m}$ and $b \neq 0$. We denote $E(\mathbb{F}_{2^m})$ to be the set of all points (x, y) with $x, y \in \mathbb{F}_{2^m}$ that satisfy the Eq. (1) together with the point at infinity \mathcal{O} . $E(\mathbb{F}_{2^m})$ forms an abelian group under the “+” operation. The identity of the abelian group is the point at infinity \mathcal{O} . Point addition can be computed by the chord and tangent method.

One of the main advantages of using binary elliptic curves over prime elliptic curves is that squaring is a linear operation in binary fields. This is the reason for having a low squaring to multiplication (\mathbf{S}/\mathbf{M}) ratio in binary fields, and it is close to a free operation. In prime fields, the \mathbf{S}/\mathbf{M} ratio is higher, and it is close to the cost of one multiplication operation [16]. It is noted that the most expensive operation for binary fields as well as prime fields is the inversion operation. The ratio (\mathbf{I}/\mathbf{M}) can be quite big (e.g. 8). One solution for reducing the cost of the inversion operation is to use projective coordinates over affine coordinates. We shall choose to derive efficient formulas in λ -projective coordinate systems as it has been proved to be better than other projective coordinates as Table 2 shows.

Another solution for reducing the cost of the inversion operation is to use efficient formulas. Efficient formulas in affine coordinates are based on the idea of trading an inversion with multiplication operations for faster performance.

Several efficient formulas for Weierstrass equation in affine coordinates have been proposed. In our case, we emphasize the $3P$ and $5P$ efficient formulas since these formulas are frequently used with MBNS methods as discussed in the next section. For affine coordinate systems, the $3P$ formula given in [32] has a very small cost of $1\mathbf{I}+5\mathbf{M}+2\mathbf{S}$ and seems hard to be further improved. In the first subsection, we are able to derive an efficient affine formula for $5P$. For λ -projective coordinate systems, fast formulas for $3P$ and $5P$ seem not available in the literature. In the second subsection of the paper, we develop efficient λ -projective coordinate formulas for $3P$ and $5P$, and these formulas will be incorporated later into MBNS methods to achieve a greater efficiency for scalar multiplication operations.

2.1 A $5P$ Formula in Affine Coordinates

As mentioned earlier, an efficient $5P$ formula under affine coordinate systems for binary elliptic curves has been proposed in [27], with a cost of $1\mathbf{I}+13\mathbf{M}+5\mathbf{S}$ [27]. We propose an improved efficient computation of $5P$ in affine coordinate system. The precise formula of $5P$ is presented by the following theorem.

Theorem 1. *Let $P = (x_P, y_P) \in E(\mathbb{F}_{2^m})$ and $6P \neq \mathcal{O}$. Set*

$$\begin{cases} \alpha = x_P^4 + x_P^3 + b \\ \beta = \alpha^2 + x_P^2(x_P^4 + b) \\ \gamma = \alpha^2(x_P^4 + b) + x_P^3\beta \end{cases} .$$

Then $5P = (x_{5P}, y_{5P})$ is given by

$$\begin{aligned} x_{5P} &= x_P + \frac{x_P^3\beta}{\gamma} + \left(\frac{x_P^3\beta}{\gamma}\right)^2 \\ y_{5P} &= y_P + x_P + (x_{5P} + x_P)\left(\frac{x_P^3\beta}{\gamma} + x_P^2 + a\right) + \frac{x_P\beta\alpha^2(\beta+(x_P^4+b)(x_P^4+b+y_P^2+x_P^2))}{\gamma^2}. \end{aligned}$$

- Remark 1.*
1. The proof of the Theorem 1 is presented in Appendix A.1.
 2. Our $5P$ efficient formula in affine coordinates costs $1\mathbf{I}+11\mathbf{M}+6\mathbf{S}$. Operation counts for our $5P$ efficient formula are given in Table 1. The least costly way for computing $5P$ without using such a formula is through $4P + P$ and this way costs $2\mathbf{I}+8\mathbf{M}+6\mathbf{S}$. With our $5P$ efficient formula, we trade $1\mathbf{I}$ with $3\mathbf{M}$ for a faster performance. Our $5P$ formula in affine coordinates saves $2\mathbf{M}$ (but with an extra \mathbf{S} whose cost is low for binary fields) over the proposed $5P$ efficient formula in [27].

2.2 $3P$ and $5P$ Formulas in λ -Projective Coordinates

The λ -coordinates system is introduced in [28] for elliptic curves over binary fields. λ -coordinates represent affine point $(x, y) \in E(\mathbb{F}_{2^m})$ by (x, λ) where $\lambda = x + \frac{y}{x}$. λ -coordinates represent a projective point by (X, L, Z) and $Z \neq 0$. λ -affine point (x_P, λ_P) is converted to λ -projective point (X_P, L_P, Z_P) by using

Table 1. Operation counts for our $5P$ in affine coordinates

Computing term	Operation counts
$\alpha = x_P^4 + x_P^3 + b$	1M+2S
$\beta = \alpha^2 + x_P^2(x_P^4 + b)$	1M+1S
$\gamma = \alpha^2(x_P^4 + b) + x_P^3\beta$	2M
$x_{5P} = x_P + \frac{x_P^3\beta}{\gamma} + \left(\frac{x_P^3\beta}{\gamma}\right)^2$	1I+1M+1S
$y_{5P} = y_P + x_P + (x_{5P} + x_P)\left(\frac{x_P^3\beta}{\gamma} + x_P^2 + a\right)$	1M
$+ \frac{x_P\beta\alpha^2(\beta+(x_P^4+b)(x_P^4+b+y_P^2+x_P^2))}{\gamma^2}$	5M+2S
	1I+11M+6S

the relation $(x_P, \lambda_P) = (\frac{X_P}{Z_P}, \frac{L_P}{Z_P})$. This representation for λ -coordinates led to an efficient $P+Q$ formula. The Weierstrass equation for λ -projective coordinates is given in [28] by

$$(L^2 + LZ + aZ^2)X^2 = X^4 + bZ^4.$$

The authors in [28] presented $2P, P + Q$, and $2Q + P$ formulas for λ -coordinates system. In this subsection, we derive efficient formulas for $3P$ and $5P$ in λ -projective coordinates.

Theorem 2. *Let $P = (X_P, L_P, Z_P) \in E(\mathbb{F}_{2^m})$. Then $3P = (X_{3P}, L_{3P}, Z_{3P})$ using λ -projective coordinates is given by*

$$\begin{aligned} T &= L_P^2 + L_P Z_P + aZ_P^2 \\ A &= (T + X_P Z_P)^2 \\ B &= T Z_P^2 + A \\ X_{3P} &= X_P Z_P B^2 \\ Z_{3P} &= Z_P^2 AB \\ L_{3P} &= T(A + B)^2 + (L_P Z_P + Z_P^2)AB. \end{aligned}$$

Remark 2. 1. The proof of Theorem 2 is presented in Appendix A.2.
 2. The cost of our $3P$ efficient formula in λ -projective coordinates, as Table 3 shows, is $8M+1M_a+5S$. The least costly way for computing $3P$ in λ -projective coordinates without the $3P$ efficient formula is through $2P + P$ with cost $15M+1M_a+6S$. With our concise $3P$ formula, we save $7M$ over $2P + P$. Our $3P$ formula saves $3M$ over the proposed $3P$ efficient formula in LD-projective coordinates in [31]. It saves $6M$ over the proposed $3P$ efficient formula in Jacobian projective coordinates in [8]. Table 2 compares the cost of $3P$ in different coordinate systems over binary fields.

Table 2. The cost for efficient formulas in different projective coordinates over binary fields

	λ -projective	LD projective	Jacobian projective
$2P$	$4M+1M_a+4S$ [28]	$4M+1M_a+4S$ [20]	$4M+1M_b+5S$ [1]
$P + Q$	$11M+2S$ [28]	$13M+4S$ [20]	$15M+1M_a+3S$ [1]
$3P$	$8M+1M_a+5S$ (<i>this work</i>)	$10M+2M_a+7S$ [31]	$13M+2M_{a,b}+7S$ [8]
$5P$	$13M+1M_a+8S$ (<i>this work</i>)	N/A	N/A

Table 3. Operation counts for our $3P$ in λ -projective coordinates

Computing term	Operation counts
$T = L_P^2 + L_P Z_P + aZ_P^2$	$1M+1M_a+2S$
$A = (T + X_P Z_P)^2$	$1M+1S$
$B = TZ_P^2 + A$	$1M$
$X_{3P} = X_P Z_P B^2$	$1M+1S$
$Z_{3P} = Z_P^2 AB$	$2M$
$L_{3P} = T(A + B)^2 + (L_P Z_P + Z_P^2)AB$	$2M+1S$
	$8M+1M_a+5S$

Theorem 3. Let $P = (X_P, L_P, Z_P) \in E(\mathbb{F}_{2^m})$. Then $5P = (X_{5P}, L_{5P}, Z_{5P})$ using λ -projective coordinates is given by

$$\begin{aligned}
 T &= L_P^2 + L_P Z_P + aZ_P^2 \\
 A &= (T + X_P Z_P)^2 \\
 B &= TZ_P^2 + A \\
 C &= (T(A + B))^2 + AB^2 \\
 D &= A^2 B + AB^2 + C \\
 X_{5P} &= X_P Z_P D^2 \\
 Z_{5P} &= Z_P^2 CD \\
 L_{5P} &= T(C + D)^2 + (L_P Z_P + Z_P^2)CD + Z_P^2 (AB)^3.
 \end{aligned}$$

Remark 3. 1. The proof of Theorem 3 is presented in Appendix A.3.
 2. The cost of our $5P$ efficient formula in λ -projective coordinates, as Table 4 shows, is $13M+1M_a+8S$. The least costly way for computing $5P$ in λ -projective coordinates without the $5P$ efficient formula is through $4P + P$ with cost $19M+2M_a+10S$. With our $5P$ efficient formula, we save $6M+1M_a$ over $4P + P$ in λ -projective coordinates. To the best of our knowledge, this is the first proposed $5P$ efficient formula in projective coordinates over binary fields, and it is the most efficient $5P$ formula for binary elliptic curves.

Table 4. Operation counts for our $5P$ in λ -projective coordinates

Computing term	Operation counts
$T = L_P^2 + L_P Z_P + a Z_P^2$	1M+1M _a +2S
$A = (T + X_P Z_P)^2$	1M+1S
$B = T Z_P^2 + A$	1M
$C = (T(A + B))^2 + AB^2$	2M+2S
$D = AB^2 + A^2 B + C$	1M +1S
$X_{5P} = X_P Z_P D^2$	1M+1S
$Z_{5P} = Z_P^2 C D$	2M
$L_{5P} = T(C + D)^2 + (L_P Z_P + Z_P^2) C D + Z_P^2 A B^2 A^2 B$	4M+1S
	13M+1M _a +8S

3 MBNS Methods

The simplest and most studied form of the MBNS is the DBNS with $\{2, 3\}$ -integers. A positive integer n is represented in the DBNS with $\{2, 3\}$ -integers in the form of

$$n = \sum_{i=1}^l s_i 2^{a_i} 3^{b_i}$$

where $a_i, b_i \geq 0, s_i \in \{-1, +1\}$, and l is the length of the expansion. The MBNS with $\{2, 3, 5\}$ -integers is a natural extension to the DBNS with $\{2, 3\}$ -integers. A positive integer n in the MBNS with $\{2, 3, 5\}$ -integers is represented by

$$n = \sum_{i=1}^l s_i 2^{a_i} 3^{b_i} 5^{c_i}$$

where $a_i, b_i, c_i \geq 0, s_i \in \{-1, +1\}$, and l is the expansion length. An MBNS expansion for integer n always exists, but it is not unique [8]. What is important to ECC is the property that under MBNS, an integer n has a short average expansion length compared to that of its single-base average expansion length, hence it minimizes the total number of point addition during the point multiplication operation.

In application, when an integer n is represented in MBNS, it has to be represented as a multi-base chain for efficiency reasons. The double-base chain with $\{2, 3\}$ -integers is decreasing sequences of the exponents a_i and b_i such that $a_1 \geq a_2 \geq \dots \geq a_l \geq 0$ and $b_1 \geq b_2 \geq \dots \geq b_l \geq 0$. The highest exponents term $2^{a_{max}} 3^{b_{max}}$ of a double-base chain is called a leading factor. The leading factor and the expansion length of a double-base chain determine the total number of operations. Thus, they have an important role for minimizing the total number of operations.

Doche in [9] defines an optimal double-base chain with $\{2, 3\}$ -integers by the following three requirements. It represents given integer n . It has a leading

factor that divides given $2^{a_{max}} 3^{b_{max}}$. It has minimum length. For example, let $n = 935811$ and $2^{20} 3^{13}$ is given. Then these chains

$$\begin{aligned} n &= 2^{12} 3^5 - 2^8 3^5 + 2^5 3^4 + 2^5 3 + 3 \\ n &= 2^7 3^8 + 2^7 3^6 + 2^5 3^4 + 2^5 3 + 3 \\ n &= 2^4 3^{10} - 2^2 3^7 - 3^5 + 3^3 - 3^2 \end{aligned}$$

are optimal for the following reasons. They have leading factors that divide the given $2^{20} 3^{13}$. The length 5 is the shorter double-base chain with $\{2, 3\}$ -integers that represents n according to the enumeration approach in [9].

Converting integer n to a double-base chain that has an optimal length on-the-fly is still an open problem [3,9]. However, efforts were made to propose methods that convert integer n to a shorter double-base chain. One of the earliest methods that converts an integer n to a double-base chain is the greedy method with restricted exponents [8]. The multi-base NAF was proposed in [23], and it is a generalization of the single-base NAF method. The ternary/binary method was proposed in [7] as an efficient scalar multiplication method that outperforms the NAF method. The tree-based method is generalized of the ternary/binary method as proposed in [10].

3.1 Experiments

Our goal in these experiments is to compare the MBNS methods with the NAF method in λ -coordinates. The tested MBNS methods are the greedy, the ternary/binary, the multi-base NAF, and the tree-based [7,8,10,23]. Our concise $3P$ and $5P$ formulas in λ -coordinates are utilized in all the tested methods. We denote $(2,3)$ greedy to be the greedy method with restricted exponents in terms of $\{2, 3\}$ -integers [8]. $(2,3)$ NAF is the multi-base NAF method with $\{2, 3\}$ -integers, and $(2,3,5)$ NAF is the multi-base NAF method with $\{2, 3, 5\}$ -integers [23]. $(2,3)$ tree is the tree-based method with $\{2, 3\}$ -integers, $(2,3,5)$ tree is the tree-based method with $\{2, 3, 5\}$ -integers, and B is the bound size [10].

The environment specifications are in the following descriptions. We used C programming language with GNU C Compiler (GCC) version 4.2. We used Intel Core i7 processor with speed 2.3 GHz. We utilized the binary field operations including: squaring, fast reduction modulo, Extended Euclidean inversion, and right-to-left comb multiplication in [17]. We used GNU Multiple Precision (GMP) library version 6.1 to generate random integers of different sizes [35]. For better accuracy, we recorded the average after trying 1000 random integers in each reading result. We used the NIST binary elliptic curves B-283, B-409, and B-571 [34].

Tables 5 and 6 show that the tested MBNS methods with our efficient formulas succeed in outperforming the NAF method. Table 5 shows the greedy, the ternary/binary, the multi-base NAF, and the tree-based methods speed up to 10%, 8%, 12%, and 15% over the NAF method. These speed-up results in Table 5 are achieved by comparing only the total number of multiplications with the

Table 5. Theoretical comparison between NAF and MBNS methods in λ -coordinates

	B-283			B-409			B-571		
	l	m	%	l	m	%	l	m	%
NAF	94.77	2173.13		136.57	3136.67		190.77	4381.11	
(2, 3)greedy	64.72	1945.81	10.46	92.66	2810.05	10.41	128.32	3925.59	10.39
Ternary/binary	65.03	1990.21	8.42	93.92	2883.57	8.06	130.53	4028.01	8.05
(2, 3)NAF	67.89	1960.33	9.79	98.09	2834.3	9.63	136.7	3956.86	9.68
(2, 3, 5)NAF	57.77	1903.77	12.39	83.56	2752.62	12.24	116.81	3846.46	12.2
(2, 3)tree $_{B=1}$	61.52	1923.25	11.49	88.39	2779.69	11.38	123.43	3889.15	11.22
(2, 3, 5)tree $_{B=1}$	50.81	1869.46	13.99	73.07	2707.51	13.68	101.81	3784.64	13.61
(2, 3, 5)tree $_{B=2}$	47.89	1839.99	15.32	68.85	2662.99	15.1	95.79	3723.41	15.01

l : The average length of the scalar expansion.
 m : The average of the total number of multiplications.
 %: The speed-up percentage in term of m .

Table 6. Running time comparison between NAF and MBNS methods in λ -coordinates

	B-283		B-409		B-571	
	Time in ms	%	Time in ms	%	Time in ms	%
NAF	32.31		78.96		198.69	
(2, 3)greedy	29.83	7.67	72.87	7.71	184.24	7.27
Ternary/binary	29.23	9.53	71.73	9.15	179.55	9.63
(2, 3)NAF	29.17	9.71	70.57	10.62	177.03	10.9
(2, 3, 5)NAF	28.16	12.84	69.03	12.57	173.21	12.82
(2, 3)tree $_{B=1}$	28.43	12.01	69.86	11.52	174.62	12.11
(2, 3, 5)tree $_{B=1}$	27.83	13.86	68.52	13.22	171.54	13.66
(2, 3, 5)tree $_{B=2}$	27.36	15.32	66.81	15.38	168.11	15.39

NAF method. It does not consider the cost of converting integer n to a multi-base chain. The conversion cost may affect the overall performance for some methods. The running time test in Table 6 considers the cost of converting n to a multi-base chain. Table 6 shows the running time of the greedy, the ternary/binary, the multi-base NAF, and the tree-based methods are up to 7%, 9%, 12%, and 15% faster than the NAF method. It shows only the running time of the greedy method has less percentage of improvement than the comparison test in Table 5. It implies converting integer n to a multi-base chain in the greedy method has a higher cost than the ternary/binary, the multi-base NAF, and the tree-based methods.

Table 5 also shows if a method has a shorter expansion length, that does not guarantee it has a lesser number of multiplications. For example, let $n = 1118848774838$, the ternary/binary method returns this chain that represents n as

$$2^{16}3^{15} + 2^{15}3^{14} + 2^{14}3^{13} - 2^{13}3^{12} - 2^{10}3^9 + 2^93^8 - 2^83^7 + 2^73^4 - 2^33^3 + 2^23 + 2.$$

Table 7. The cost of greedy method with different values of (a_{max}, b_{max}) in λ -coordinates

B-283				B-409				B-571			
a_{max}	b_{max}	l	m	a_{max}	b_{max}	l	m	a_{max}	b_{max}	l	m
140	91	75.95	2100.73	205	129	108.02	3020.44	285	181	153.14	4251.62
160	79	65.42	1964.21	230	113	94.14	2841.52	325	155	130.04	3954.89
170	72	64.72	1945.81	245	104	92.66	2810.05	345	143	129.09	3923.85
180	65	66.29	1951.83	260	95	95.16	2822.14	365	130	133.01	3945.73
200	53	71.22	1985.8	290	75	103.78	2884.58	405	104	144.96	4034.8
220	40	77.06	2028.37	320	57	111.42	2938.09	445	79	155.84	4113.04
240	27	82.43	2066.38	350	38	119.57	2996.4	485	54	166.89	4192.71
260	15	87.83	2105.48	380	18	128.26	3059.8	525	28	177.83	4270.8

l : The average length of the scalar expansion.
 m : The average of the total number of multiplications.

The length of this chain is 11, and it costs in λ -coordinates $16 \times 4 + 15 \times 8 + 10 \times 11 = 294M$. See Table 2 for the cost of $P + Q$, $2P$, and $3P$ efficient formulas in λ -coordinates. The multi-base NAF with $\{2, 3\}$ -integers returns this chain that represents n as

$$2^{32}3^5 + 2^{30}3^4 - 2^{27}3^4 - 2^{25}3^3 - 2^{21}3^2 - 2^{19}3^2 - 2^{14}3^2 - 2^{11}3^2 - 2^93 + 2^63 - 2^3 - 2.$$

The length of this chain is 12, and it costs in λ -coordinates 289M. This example explains, as Table 5 shows, the ternary/binary method has a shorter average length than the multi-base NAF method with $\{2, 3\}$ -integers. However, the multi-base NAF with $\{2, 3\}$ -integers method has a lesser average number of multiplications than the ternary/binary method. In Table 5, the tree-based succeeds in generating a shorter average length than other tested MBNS methods. The tree-based method with bound size $B = 1$ does not always produce an optimal chain. For example, let $n = 1118848774838$, the tree-based with $\{2, 3\}$ -integers and $B = 1$ returns this chain that represents n as

$$2^{21}3^{12} + 2^{18}3^9 - 2^{17}3^8 + 2^{14}3^7 + 2^83^7 + 2^73^4 - 2^33^3 + 2^23 + 2.$$

The length of this chain is 9, and it costs in λ -coordinates 268M. According to the enumeration approach in [9], the optimal chain with $\{2, 3\}$ -integers that represents n is

$$2^{21}3^{12} + 2^{13}3^{12} - 2^{13}3^7 + 2^83^7 + 2^73^4 - 2^33^3 + 2^23 + 2.$$

The optimal chain length is 8, and it costs in λ -coordinates 257M.

Table 5 also shows the greedy method with $\{2, 3\}$ -integers has a shorter average length and a lesser average number of multiplications than the ternary/binary and the multi-base NAF method with $\{2, 3\}$ -integers. However, the greedy method result in Table 5 does not consider the conversion cost nor the effort to select the best upper bound (a_{max}, b_{max}) as Table 7 shows. In Table 7, we tried values from $\frac{\log_2 n}{2}$ to $\log_2 n$ for a_{max} such that $a_{max} + b_{max} \log_2 3 \approx \log_2 n$.

We selected $(a_{max} = 170, b_{max} = 72)$, $(a_{max} = 245, b_{max} = 104)$, and $(a_{max} = 345, b_{max} = 143)$ for the irreducible polynomials of degree 283, 409, and 571 respectively. We used a line search algorithm to find the best approximation for integer n in term of a $\{2, 3\}$ -integer [32]. We did not find it a practical to use a look-up table as proposed in [11]. The look-up table contains off-line pre-computation for all integers n and their corresponding in term of a $\{2, 3\}$ -integer.

4 The Window TNAF for Koblitz Curves

Koblitz introduced in [19] an efficiently computable endomorphism with a special class of elliptic curves. Koblitz defined the special class of curves E_a over binary fields \mathbb{F}_{2^m} by

$$E_a : y^2 + xy = x^3 + ax^2 + 1, \quad (2)$$

where $a \in \{0, 1\}$. We denote $E_a(\mathbb{F}_{2^m})$ to be the set of all points (x, y) that satisfy the Eq. (2), plus the point at infinity \mathcal{O} . The properties of Koblitz curves allow a scalar multiplication to use the *Frobenius* map instead of point doubling. The *Frobenius* map $\tau : E_a(\mathbb{F}_{2^m}) \rightarrow E_a(\mathbb{F}_{2^m})$ is defined by $\tau(x, y) = (x^2, y^2)$, $\tau(\mathcal{O}) = \mathcal{O}$. One property of E_a is that $\tau^2(P) + 2P = \mu\tau(P)$, for all $P \in E_a(\mathbb{F}_{2^m})$, where $\mu = (-1)^{1-a}$. This means τ can be considered to be a complex number that satisfies $\tau^2 + 2 = \mu\tau$. By solving $\tau^2 - \mu\tau + 2 = 0$, there is a choice for $\tau = \frac{\mu + \sqrt{-7}}{2}$. Solinas in [29] showed the window TNAF method can be used with Koblitz curves. It needs to perform an online pre-computation for $2^{w-2} - 1$ points where w is the selected window width.

4.1 A 3P Formula for the Optimal Pre-computation of Window TNAF

Trost and Xu in [30] established an optimal arrangement setting for the pre-computed points of window TNAF. The optimal pre-computation of window TNAF, as Table 8 shows, costs one point addition and two evaluations of τ at most for each pre-computed point. Also in [30] they proposed improvements for the optimal pre-computation of window TNAF by replacing point additions with the efficient formulas in λ -coordinates. The efficient formulas are for the pre-computed points in the forms of $P - \mu\tau(P)$, $P + \mu\tau(P)$, and $P - \tau^2(P)$.

Our contribution in this section has two parts. Frist, we propose a 3P efficient formula that can be used together with the already proposed efficient formulas for further speed-up of the optimal pre-computation of window TNAF. Secondly, we conduct experiments to measure the achieved improvement for the optimal pre-computation of window TNAF by using these proposed efficient formulas.

Table 8. The optimal pre-computation of window TNAF when $a = 0$.

Width	Pre-computed points		
4	$Q_3 = -P + \tau^2 P$	$Q_5 = -P - \tau P$	$Q_7 = P - \tau P$
5	$Q_3 = -P + \tau^2 P$	$Q_5 = -P - \tau P$	$Q_7 = P - \tau P$
	$Q_9 = Q_3 - \tau P$	$Q_{11} = Q_5 - \tau P$	$Q_{13} = Q_7 - \tau P$
	$Q_{15} = -Q_{11} + \tau P$		
6	$Q_{29} = P - \tau^2 P$	$Q_3 = Q_{29} - \tau P$	$Q_{31} = Q_3 - \tau^2 P$
	$Q_5 = Q_{31} - \tau P$	$Q_7 = -Q_{31} - \tau P$	$Q_9 = -Q_{29} - \tau P$
	$Q_{27} = P + \tau P$	$Q_{11} = -Q_{27} - \tau P$	$Q_{25} = -P + \tau P$
	$Q_{13} = -Q_{25} - \tau P$	$Q_{15} = -Q_{11} + \tau P$	$Q_{17} = -Q_9 + \tau P$
	$Q_{19} = -Q_7 + \tau P$	$Q_{21} = -Q_{17} - \tau P$	$Q_{23} = -Q_3 + \tau P$

When $a = 1$, Q_j can be obtained by changing only the sign of τ .

Recall that the proposed efficient formulas for the pre-computed points in the forms of $P - \mu\tau(P)$ and $P + \mu\tau(P)$ are given in [30] by

$$\begin{aligned}
 A &= X_P(X_P + Z_P)^2 \\
 B &= X_P^4 + X_P Z_P + Z_P^4 \\
 X_{P-\mu\tau(P)} &= (X_P + Z_P)^4 \\
 L_{P-\mu\tau(P)} &= L_P A + X_P^3 Z_P \\
 Z_{P-\mu\tau(P)} &= Z_P A \\
 X_{P+\mu\tau(P)} &= B^2 \\
 L_{P+\mu\tau(P)} &= X_P^7 Z_P + L_P A B \\
 Z_{P+\mu\tau(P)} &= Z_P A B.
 \end{aligned}$$

The pre-computed point in the form of $P - \tau^2(P)$ can be computed by letting $Q = P + \mu\tau(P)$. Then, we have $P - \tau^2(P) = Q - \mu\tau(Q)$. The pre-computed point $3P$ can be computed by Theorem 2 and it can be recognized in the optimal pre-computation of window TNAF by the following proposition.

Proposition 1. *Let $P = (x_P, y_P) \in E_a(\mathbb{F}_{2^m})$ for Koblitz curve E_a . Then*

$$3P = P - \tau^2(P) + \mu\tau(P).$$

Proof. We know $(\tau^2 + 2)P = \mu\tau(P)$ for all $P \in E_a(\mathbb{F}_{2^m})$. It means $2P = \mu\tau(P) - \tau^2(P)$. It implies $2P + P = \mu\tau(P) - \tau^2(P) + P$. Thus, $3P = P - \tau^2(P) + \mu\tau(P)$.

For example, consider the optimal pre-computation of window TNAF with $w = 6$. Then, the pre-computed point Q_3 can be computed by the $3P$ efficient formulas. To explain, $Q_3 = Q_{29} + \mu\tau(P) = P - \tau^2(P) + \mu\tau(P) = 3P$.

As mentioned earlier, the improvement of the optimal pre-computation of window TNAF is achieved by replacing point additions with the efficient formulas. Recall that the point addition in λ -projective coordinates costs $11\mathbf{M}+2\mathbf{S}$. The efficient formulas for the pre-computed point in the form of $P - \mu\tau(P)$ costs $5\mathbf{M}+3\mathbf{S}$. The efficient formulas for the pre-computed point in the form of $P + \mu\tau(P)$ costs $7\mathbf{M} + 5\mathbf{S}$. The efficient formulas for the pre-computed point $3P$ costs $8\mathbf{M}+6\mathbf{S}$. Thus, the pre-computed points for the above cost less than point addition in λ -projective coordinates.

4.2 Experiments

The goal for these experiments are to measure the improvement for the optimal pre-computation of window TNAF with the efficient formulas. We replaced the pre-computed points in the forms of $P - \mu\tau(P)$, $P + \mu\tau(P)$, $P - \tau^2(P)$, and $3P$ of the optimal pre-computation of window TNAF with the efficient formulas. For simplicity, we denote OPT to be the optimal pre-computation of window TNAF without the efficient formulas. We denote OPT+ to be the optimal pre-computation of window TNAF with the efficient formulas. We used two tests to measure the performance of OPT and OPT+. In the first test, we compare OPT and OPT+ in terms of the number of multiplications, as Table 9 shows. In the second test, we did a software implementation, as Table 10 shows, for OPT and OPT+. We used the NIST Koblitz curves K-283, K-409, and K-571 [34]. The environment specifications for these experiments are similar to the experiments in Sect. 3.1.

Table 9 shows OPT+ speeds up to 48%, 24% and 11% over OPT for window width 4, 5 and 6 respectively. In Table 9, we counted the number of inversions, multiplications of OPT and OPT+ in different window width. We converted an inversion to multiplication based on the ratio \mathbf{I}/\mathbf{M} assumption. We presented two cases for the \mathbf{I}/\mathbf{M} ratio in affine coordinates. The first case is the number of multiplications for OPT when the \mathbf{I}/\mathbf{M} ratio = 5. The second case is the number of multiplications for OPT when the \mathbf{I}/\mathbf{M} ratio = 8. A squaring operation was ignored in this method since squaring is almost a free operation over binary fields.

Table 10 shows the running time of OPT+ in λ -projective coordinates performs faster than OPT. However, the percentage of improvement is different for

Table 9. Theoretical comparison in terms of the number of multiplications

	Affine coordinates		λ -projective coordinates		
	OPT: $\mathbf{I}/\mathbf{M}=5$	OPT: $\mathbf{I}/\mathbf{M}=8$	OPT	OPT+	%
$w = 4$	21	30	33	17	48.48
$w = 5$	49	70	77	58	24.67
$w = 6$	105	150	165	146	11.51

OPT: The optimal pre-computation without the efficient formulas.

OPT+: The optimal pre-computation with the efficient formulas.

Table 10. Running time comparison between OPT and OPT+

Coordinates		Affine	λ -projective		
		OPT	OPT	OPT+	%
K-283	$w = 4$	0.58 ms	0.517 ms	0.281 ms	45.64
	$w = 5$	1.34 ms	1.113 ms	0.901 ms	19.04
	$w = 6$	2.92 ms	2.401 ms	2.188 ms	8.87
K-409	$w = 4$	1.05 ms	0.844 ms	0.461 ms	45.37
	$w = 5$	2.54 ms	1.615 ms	1.993 ms	18.96
	$w = 6$	5.34 ms	4.101 ms	3.694 ms	9.92
K-571	$w = 4$	1.94 ms	1.553 ms	0.834 ms	46.29
	$w = 5$	4.55 ms	3.565 ms	2.893 ms	18.84
	$w = 6$	9.74 ms	7.488 ms	6.794 ms	9.26

OPT: The optimal pre-computation without the efficient formulas.

OPT+: The optimal pre-computation with the efficient formulas.

each window width. It shows OPT+ in λ -projective coordinates gives up to a 46%, 19%, and 9% speed-up over OPT if the selected window width is 4, 5, and 6 respectively.

5 Conclusion

In this paper, we present the most efficient $3P$ and $5P$ formulas for binary elliptic curves. We are the first to derive efficient formulas for $3P$ and $5P$ in λ -projective coordinates. We also derived the most efficient $5P$ formula in affine coordinates. Our efficient formulas have an important role in speeding up scalar multiplication operations based on MBNS. We investigated the following MBNS methods: the greedy, the ternary/binary, the multi-base NAF, and the tree-based. We conducted performance comparison tests to these methods using our formulas with respect to the expansion length, the total number of multiplications, and the running time. The total number of multiplications test shows the greedy, the ternary/binary, the multi-base NAF, and the tree-based methods speed up to 10%, 8%, 12%, and 15% over the NAF method. Our running time test shows that the greedy method has a lower percentage of improvement since this test considers the time of converting integer n to a multi-base chain. It implies the greedy method has a higher conversion cost than other MBNS methods.

We proposed a $3P$ efficient formula for the optimal pre-computation of window TNAF for Koblitz curves. Our $3P$ formula can be used with the already proposed efficient formulas to the pre-computed points in the forms of $P - \mu\tau(P)$, $P + \mu\tau(P)$, and $P - \tau^2(P)$. Our experiments show the optimal pre-computation of window TNAF using the efficient formulas speed up to 48%, 24%, and 11% if the used window width is 4, 5, and 6 respectively.

A Appendix: Proofs

A.1 Theorem 1

Proof. Let $P = (x_P, y_P) \in E(\mathbb{F}_{2^m})$ and $6P \neq \mathcal{O}$. Otherwise, if $6P = \mathcal{O}$ then $5P = -P$. We shall prove Theorem 1 by the fact

$$(x_{5P}, \lambda_{5P}) = (x_{2P}, \lambda_{2P}) + (x_{3P}, \lambda_{3P}).$$

By using the $P + Q$ λ -affine formula given in [28], we have

$$x_{5P} = \frac{x_{3P}x_{2P}}{(x_{3P} + x_{2P})^2}(\lambda_{3P} + \lambda_{2P}). \quad (3)$$

$$\lambda_{5P} = \frac{x_{3P}(x_{5P} + x_{2P})^2}{x_{5P}x_{2P}} + \lambda_{2P} + 1. \quad (4)$$

We apply $x_{3P} = x_P + \frac{x_P^3}{\alpha} + (\frac{x_P^3}{\alpha})^2$ and $x_{2P} = \frac{x_P^4+b}{x_P^2}$ in Eq. (3). We have

$$x_{5P} = \frac{x_P^3(\alpha^2 + x_P^2(x_P^4 + b))\alpha^2(x_P^4 + b)}{(\alpha^2(x_P^4 + b) + x_P^3(\alpha^2 + x_P^2(x_P^4 + b)))^2}(\lambda_{3P} + \lambda_{2P}) \quad (5)$$

$$= \frac{x_P^3\beta\alpha^2(x_P^4 + b)}{\gamma^2}(\lambda_{3P} + \lambda_{2P}) \quad (6)$$

We note that

$$\lambda_{3P} + \lambda_{2P} = \frac{x_P\gamma^2}{x_P^3\beta\alpha^2(x_P^4 + b)} + 1. \quad (7)$$

By applying Eq. (7) in Eq. (6), we have

$$x_{5P} = x_P + \frac{x_P^3\beta\alpha^2(x_P^4 + b)}{\gamma^2} \quad (8)$$

$$= x_P + \frac{x_P^3\beta}{\gamma} + \left(\frac{x_P^3\beta}{\gamma}\right)^2. \quad (9)$$

We have derived x_{5P} . Next, we want to derive y_{5P} . From Eq. (4), we have

$$\lambda_{5P} = \frac{x_{3P}}{x_{2P}}x_{5P} + \frac{x_{3P}x_{2P}}{x_{5P}} + \lambda_{2P} + 1. \quad (10)$$

We apply Eq. (10) to the fact $y_{5P} = x_{5P}(\lambda_{5P} + x_{5P})$. We have

$$y_{5P} = x_{5P}\left(\frac{x_{3P}}{x_{2P}}x_{5P} + \lambda_{2P} + 1 + x_{5P}\right) + x_{3P}x_{2P}. \quad (11)$$

We apply x_{3P}, x_{2P} , and $\lambda_{2P} = \frac{x_P^4}{x_P^4+b} + \lambda_P^2 + a + 1$ in Eq. (11). We have

$$\begin{aligned} y_{5P} &= x_{5P} \left(\frac{x_P^3 \beta}{\alpha^2(x_P^4+b)} x_{5P} + \frac{x_P^4}{x_P^4+b} + \frac{y_P^2}{x_P^2} + x_P^2 + a + x_{5P} \right) + \frac{(x_P^4+b)\beta}{x_P \alpha^2} \\ &= x_{5P} \left(\left(\frac{x_P^3 \beta}{\gamma} \right)^2 + x_P^2 + a + x_{5P} \right) + \frac{(x_P^4+b)\beta}{x_P \alpha^2} + \frac{x_P^4 \beta}{\alpha^2(x_P^4+b)} x_{5P} \\ &\quad + \frac{x_P^6 + y_P^2(x_P^4+b)}{x_P^2(x_P^4+b)} x_{5P} \\ &= y_P + x_P + (x_{5P} + x_P) \left(\left(\frac{x_P^3 \beta}{\gamma} \right)^2 + x_P^2 + a + x_{5P} + x_P \right) \\ &\quad + \frac{x_P \beta \alpha^2 (x_P^6 + y_P^2(x_P^4+b))}{\gamma^2}. \end{aligned}$$

We note that $x_P^6 = \beta + (x_P^4+b)^2 + x_P^2(x_P^4+b)$ and $\left(\frac{x_P^3 \beta}{\gamma}\right)^2 = \frac{x_P^3 \beta}{\gamma} + x_{5P} + x_P$. We have

$$y_{5P} = y_P + x_P + (x_{5P} + x_P) \left(\frac{x_P^3 \beta}{\gamma} + x_P^2 + a \right) + \frac{x_P \beta \alpha^2 (\beta + (x_P^4+b)(x_P^4+b + y_P^2 + x_P^2))}{\gamma^2}.$$

A.2 Theorem 2

Proof. We shall prove Theorem 2 by the fact

$$(x_{3P}, \lambda_{3P}) = (x_P, \lambda_P) + (x_{2P}, \lambda_{2P}). \tag{12}$$

By using the $P + Q$ λ -affine formula given in [28], we have

$$x_{3P} = \frac{x_P x_{2P}}{(x_P + x_{2P})^2} (\lambda_P + \lambda_{2P}). \tag{13}$$

$$\lambda_{3P} = \frac{x_{2P}(x_{3P} + x_P)^2}{x_{3P} x_P} + \lambda_P + 1. \tag{14}$$

We apply the relation $\lambda_P + \lambda_{2P} = \frac{(x_P+x_{2P})^2}{x_{2P}} + 1$ in Eq. (13). We have

$$x_{3P} = x_P + \frac{x_P x_{2P}}{(x_P + x_{2P})^2} \tag{15}$$

$$= \frac{x_P (x_{2P} + (x_{2P} + x_P)^2)}{(x_P + x_{2P})^2}. \tag{16}$$

We convert λ -affine point (x_P, λ_P) to λ -projective point (X_P, L_P, Z_P) by using the relation $(x_P, \lambda_P) = \left(\frac{X_P}{Z_P}, \frac{L_P}{Z_P}\right)$. Thus, the equations above become

$$x_{2P} = \frac{L_P^2 + L_P Z_P + a Z^2}{Z_P^2} = \frac{T}{Z_P^2}.$$

$$\begin{aligned}
 x_{3P} &= \frac{\frac{X_P}{Z_P} \left(\frac{T}{Z_P^2} + \frac{(T+X_P Z_P)^2}{Z_P^4} \right)}{\frac{(T+X_P Z_P)^2}{Z_P^4}} \\
 &= \frac{X_P (T Z_P^2 + (T + X_P Z_P)^2)}{Z_P (T + X_P Z_P)^2} \\
 &= \frac{X_P B}{Z_P A}. \\
 \lambda_{3P} &= \frac{\frac{T}{Z_P^2} \left(\frac{X_P B}{Z_P A} + \frac{X_P}{Z_P} \right)^2}{\frac{X_P^2 B}{Z_P^2 A}} + \frac{L_P Z_P + Z_P^2}{Z_P^2} \\
 &= \frac{T(A+B)^2}{Z_P^2 AB} + \frac{L_P Z_P + Z_P^2}{Z_P^2} \\
 &= \frac{T(A+B)^2 + (L_P Z_P + Z_P^2) AB}{Z_P^2 AB}.
 \end{aligned}$$

A.3 Theorem 3

Proof. We shall proof x_{5P} by the fact

$$(x_{5P}, \lambda_{5P}) = (x_{2P}, \lambda_{2P}) + (x_{3P}, \lambda_{3P}).$$

By using the $P + Q$ λ -affine formula given in [28], we have

$$x_{5P} = \frac{x_{2P} x_{3P}}{(x_{2P} + x_{3P})^2} (\lambda_{2P} + \lambda_{3P}). \tag{17}$$

We apply the relation $\lambda_{2P} + \lambda_{3P} = \frac{x_P(x_{2P}+x_{3P})^2}{x_{2P}x_{3P}} + 1$ to Eq. (17). We have

$$x_{5P} = x_P + \frac{x_{2P} x_{3P}}{(x_{2P} + x_{3P})^2} \tag{18}$$

$$= \frac{x_P(x_{2P} + x_{3P})^2 + x_{2P} x_{3P}}{(x_{2P} + x_{3P})^2}. \tag{19}$$

Next, we shall derive λ_{5P} by the fact

$$(x_{5P}, \lambda_{5P}) = (x_P, \lambda_P) + (x_{4P}, \lambda_{4P}).$$

By using the $P + Q$ λ -affine formula, we have

$$\lambda_{5P} = \frac{x_{4P}(x_{5P} + x_P)^2}{x_{5P} x_P} + \lambda_P + 1. \tag{20}$$

We convert λ -affine point (x_P, λ_P) to λ -projective point (X_P, L_P, Z_P) by using the relation $(x_P, \lambda_P) = (\frac{X_P}{Z_P}, \frac{L_P}{Z_P})$. Thus, the equations above become

$$\begin{aligned}
 x_{2P} &= \frac{L_P^2 + L_P Z_P + a Z^2}{Z_P^2} = \frac{T}{Z_P^2}. \\
 x_{3P} &= \frac{X_P(T Z_P^2 + (T + X_P Z_P)^2)}{Z_P(T + X_P Z_P)^2} = \frac{X_P B}{Z_P A}. \\
 x_{4P} &= \frac{L_{2P}^2 + L_{2P} T Z_P^2 + a(T Z^2)^2}{(T Z_P^2)^2} = \frac{T_2}{(T Z_P^2)^2}. \\
 x_{5P} &= \frac{\frac{X_P}{Z_P} \left(\frac{T}{Z_P} + \frac{X_P B}{Z_P A} \right)^2 + \frac{T X_P B}{Z_P^3 A}}{\left(\frac{T}{Z_P} + \frac{X_P B}{Z_P A} \right)^2} \\
 &= \frac{X_P \left((T A + X_P Z_P B)^2 + T Z_P^2 A B \right)}{Z_P (T A + X_P Z_P B)^2} \\
 &= \frac{X_P D}{Z_P C}. \\
 \lambda_{5P} &= \frac{\frac{T_2}{(T Z_P^2)^2} \left(\frac{X_P D}{Z_P C} + \frac{X_P}{Z_P} \right)^2}{\frac{X_P^2 D}{Z_P^2 C}} + \frac{L_P Z_P + Z_P^2}{Z_P^2} \\
 &= \frac{T_2 (C + D)^2}{(T Z_P^2)^2 C D} + \frac{L_P Z_P + Z_P^2}{Z_P^2} \\
 &= \frac{Z^2 T_2 (A B)^2 + (L_P Z_P + Z_P^2) C D}{Z_P^2 C D}.
 \end{aligned}$$

We note the following relations

$$\begin{aligned}
 Z_P^2 T_2 &= T(A + B)^2 + Z_P^2 A B. \\
 C &= (T A + X_P Z_P B)^2 = (T(A + B))^2 + A B^2. \\
 D &= T Z_P^2 A B + C = A^2 B + A B^2 + C.
 \end{aligned}$$

Thus, we have

$$L_{5P} = T(C + D)^2 + (L_P Z_P + Z_P^2) C D + Z_P^2 (A B)^3.$$

References

1. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Nguyen, K., Lange, T., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Chapman & Hall/CRC, Boca Raton (2005)
2. Bernstein, D.J., Birkner, P., Lange, T., Peters, C.: Optimizing double-base elliptic-curve single-scalar multiplication. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 167–182. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77026-8_13

3. Berthé, V., Imbert, L.: On converting numbers to the double-base number system. In: *Advanced Signal Processing Algorithms, Architecture and Implementations XIV*, vol. 5559, pp. 70–78 (2004)
4. Blake, I.F., Murty, V.K., Xu, G.: A note on window τ -NAF algorithm. *Inf. Process. Lett.* **95**(5), 496–502 (2005)
5. Blake, I.F., Murty, V.K., Xu, G.: Nonadjacent radix- τ expansions of integers in euclidean imaginary quadratic number fields. *Can. J. Math.* **60**, 1267–1282 (2008)
6. Blake, I.F., Seroussi, G., Smart, N.P.: *Elliptic Curves in Cryptography*. Cambridge University Press, Cambridge (1999)
7. Ciet, M., Joye, M., Lauter, K., Montgomery, P.L.: Trading inversions for multiplications in elliptic curve cryptography. *Desi. Codes Cryptogr.* **39**(2), 189–206 (2006)
8. Dimitrov, V., Imbert, L., Mishra, P.K.: The double-base number system and its application to elliptic curve cryptography. *Math. Comput.* **77**(262), 1075–1104 (2008)
9. Doche, C.: On the enumeration of double-base chains with applications to elliptic curve cryptography. In: Sarkar, P., Iwata, T. (eds.) *ASIACRYPT 2014*. LNCS, vol. 8873, pp. 297–316. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_16
10. Doche, C., Habsieger, L.: A tree-based approach for computing double-base chains. In: Mu, Y., Susilo, W., Seberry, J. (eds.) *ACISP 2008*. LNCS, vol. 5107, pp. 433–446. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70500-0_32
11. Doche, C., Imbert, L.: Extended double-base number system with applications to elliptic curve cryptography. In: Barua, R., Lange, T. (eds.) *INDOCRYPT 2006*. LNCS, vol. 4329, pp. 335–348. Springer, Heidelberg (2006). https://doi.org/10.1007/11941378_24
12. Doche, C., Kohel, D.R., Sica, F.: Double-base number system for multi-scalar multiplications. In: Joux, A. (ed.) *EUROCRYPT 2009*. LNCS, vol. 5479, pp. 502–517. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_29
13. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. In: Joux, A. (ed.) *EUROCRYPT 2009*. LNCS, vol. 5479, pp. 518–535. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_30
14. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) *CRYPTO 2001*. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_11
15. Hankerson, D., Karabina, K., Menezes, A.: Analyzing the Galbraith-Lin-Scott point multiplication method for elliptic curves over binary fields. *IEEE Trans. Comput.* **58**(10), 1411–1420 (2009)
16. Hankerson, D., López Hernandez, J., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. In: Koç, Ç.K., Paar, C. (eds.) *CHES 2000*. LNCS, vol. 1965, pp. 1–24. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44499-8_1
17. Hankerson, D., Menezes, A., Vanstone, S.: *Guide to Elliptic Curve Cryptography*. Springer, New York (2004). <https://doi.org/10.1007/b97644>
18. Koblitz, N.: CM-curves with good cryptographic properties. In: Feigenbaum, J. (ed.) *CRYPTO 1991*. LNCS, vol. 576, pp. 279–287. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_22
19. Koblitz, N.: Elliptic curve cryptosystems. *Math. Computat.* **48**(177), 203–209 (1987)

20. Lange, T.: A note on López-Dahab coordinates. Cryptology ePrint Archive, Report 2004/323 (2004). <https://eprint.iacr.org>
21. Lenstra, A., Verheul, E.: Selecting cryptographic key sizes. *J. Cryptol.* **14**, 255–293 (2001)
22. Longa, P.: Accelerating the scalar multiplication on elliptic curve cryptosystems over prime fields. Cryptology ePrint Archive, Report 2008/100 (2008). <https://eprint.iacr.org>
23. Longa, P., Gebotys, C.: Fast multibase methods and other several optimizations for elliptic curve scalar multiplication. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 443–462. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_25
24. López, J., Dahab, R.: Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 201–212. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48892-8_16
25. Méloni, N., Hasan, M.: Efficient double bases for scalar multiplication. *IEEE Trans. Comput.* **64**, 2204–2212 (2015)
26. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_31
27. Mishra, P.K., Dimitrov, V.: Efficient quintuple formulas for elliptic curves and efficient scalar multiplication using multibase number representation. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 390–406. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75496-1_26
28. Oliveira, T., López, J., Aranha, D.F., Rodríguez-Henríquez, F.: Two is the fastest prime: lambda coordinates for binary elliptic curves. *J. Cryptogr. Eng.* **4**(1), 3–17 (2014)
29. Solinas, J.A.: Efficient arithmetic on Koblitz curves. *Des. Codes Cryptogr.* **19**(2–3), 195–249 (2000)
30. Trost, W., Xu, G.: On the optimal pre-computation of window tNAF for Koblitz curves. *IEEE Trans. Comput.* **65**, 2918–2924 (2016)
31. Yasin, S., Muda, Z.: Tripling formulae of elliptic curve over binary field in Lopez-Dahab model. *J. Theor. Appl. Inf. Technol.* **75**(2), 212–217 (2015)
32. Yu, W., Kim, K.H., Jo, M.S.: New fast algorithms for elliptic curve arithmetic in affine coordinates. In: Tanaka, K., Suga, Y. (eds.) IWSEC 2015. LNCS, vol. 9241, pp. 56–64. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22425-1_4
33. Yu, W., Wang, K., Li, B., Tian, S.: Triple-base number system for scalar multiplication. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 2013. LNCS, vol. 7918, pp. 433–451. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38553-7_26
34. Digital Signature Standard (DSS): FIPS PUB. 186–4 (2013)
35. The GNU Multiple Precision Arithmetic Library. <http://www.gmpilib.org>

Field Lifting for Smaller UOV Public Keys

Ward Beullens^(✉) and Bart Preneel

imec-COSIC KU Leuven, Kasteelpark Arenberg 10 - bus 2452,
3001 Heverlee, Belgium
{ward.beullens,bart.preneel}@esat.kuleuven.be

Abstract. Most Multivariate Quadratic (MQ) signature schemes have a very large public key, which makes them unsuitable for many applications, despite attractive features such as speed and small signature sizes. In this paper we introduce a modification of the Unbalanced Oil and Vinegar (UOV) signature scheme that has public keys which are an order of magnitude smaller than other MQ signature schemes. The main idea is to choose UOV keys over the smallest field \mathbb{F}_2 in order to achieve small keys, but to lift the keys to a large extension field, where solving the MQ problem is harder. The resulting Lifted UOV signature scheme is very competitive with other post-quantum signature schemes in terms of key sizes, signature sizes and speed.

Keywords: Post-quantum cryptography · Multivariate cryptography
Signature schemes · Unbalanced oil and vinegar · Key size reduction

1 Introduction

When large scale quantum computers are built, they will be able to break nearly all public key cryptography that is being used today, including RSA [25], DSA [17] and ECC. This is because these schemes rely on the hardness of number theoretic problems such as integer factorization and finding discrete logarithms, which can be solved efficiently by Shor's Algorithm [26]. Even if it would take 10 or 20 years to build large scale quantum computers, upgrading our current systems may be very slow and some stored data requires long term protection (in particular for confidentiality). To avert a potential catastrophe, post-quantum cryptography should be designed, implemented and deployed well before large scale quantum computers are built.

During recent years, the research on post-quantum cryptography has been accelerating. One of the goals of the EU-funded PQCRYPTO project is to develop and standardize post-quantum algorithms [1]. Recently NIST, the US National Institute for Standards and Technology, has started the process of selecting post-quantum algorithms for standardization [19]. According to both PQCRYPTO and NIST, multivariate cryptography is one of the major candidates for providing post-quantum security. Multivariate cryptography is based on the hardness of some problems related to multivariate polynomials over finite

fields, such as solving multivariate polynomial equations. In general, multivariate cryptography is very fast and requires only moderate computational resources, which makes it attractive for applications in low-cost devices. However, a disadvantage of multivariate cryptography is its large public keys, which can be prohibitive for many applications. Some work in mitigating this problem in the case of the UOV and Rainbow signature schemes has been published by Petzoldt [22], who managed to reduce the key size by a factor of 8 in the case of UOV and a factor of 3 in the case of the Rainbow signature scheme. His proposal makes a small modification to the key generation algorithm and exploits the fact that a large part of the public key can be freely chosen by the user. One can then choose to generate this part using a Pseudo-Random Number Generator (PRNG), and to only store the seed for the PRNG. In this paper we introduce a new idea to reduce the size of the public keys of UOV dramatically, by lifting the public and central maps to an extension field. The new idea is compatible with the ideas of Petzoldt and together they provide public keys that are up to 10 times smaller than if we were to use only Petzoldt's modification of UOV.

Before introducing the Lifted UOV signature scheme in Sect. 5, we present an overview of the MQ problem in Sect. 2 and the UOV signature and how it was improved by Pezoldt in Sects. 3 and 4. We finish with a brief description of our software implementation in Sect. 6 and conclude in Sect. 7.

2 The MQ Problem

The security of an MQ signature scheme relies on the hardness of the MQ-problem. We give a brief discussion of the problem here.

MQ Problem. Given a quadratic polynomial map $\mathcal{P} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$ over a finite field \mathbb{F}_q , find $\mathbf{x} \in \mathbb{F}_q^n$ that satisfies $\mathcal{P}(\mathbf{x}) = \mathbf{0}$.

It is known that the MQ problem is NP-hard [18]. Therefore it is unlikely that there are (quantum) algorithms that solve the hardest instances of the MQ problem in polynomial time. The problem is also believed to be hard on average in the case $n \approx m$. Only exponential time algorithms are known to solve random instances of the problem for these parameters.

Systems with $n = m$ are called determined systems; these are the most difficult systems to solve. When $n < m$ a system is called overdetermined, and when $n > m$ the system is called underdetermined. Thomae et al. showed that finding a solution for an underdetermined system with $n = \alpha m$ can be reduced to finding a solution of a determined system with only $m + 1 - \lfloor \alpha \rfloor$ equations [27]. This means that as a system becomes more underdetermined it becomes easier to solve. This fact will become important in the security analysis of UOV.

2.1 Classical Algorithms

The best known classical algorithms to solve the MQ-problem for generic determined systems over finite fields use the hybrid approach [5,6]. This approach combines exhaustive search with Gröbner basis computations. In this approach k variables are fixed to random values and the remaining $n - k$ variables are found with a Gröbner basis algorithm such as F_4 , F_5 or XL. If no assignment to the remaining $n - k$ variables exists that solves the system, the procedure starts again with a different guess for the first k variables. We require on average roughly q^k Gröbner basis computations until a solution is found. As a result, the optimal value of k decreases as q increases. The complexity of computing a Gröbner basis for a system of polynomials depends critically on the degree of regularity (d_{reg}) of that system. Though it is of little importance to the rest of the paper, we refer to Bardet [2] for a precise definition of the degree of regularity. The complexity of the F_5 algorithm is given by

$$C_{F_5}(n, d_{reg}) = O \left(\binom{n + d_{reg}}{d_{reg}}^\omega \right),$$

where $2 \leq \omega < 3$ is the constant in the complexity of matrix multiplication. Therefore the complexity of the hybrid approach is

$$C_{\text{Hybrid}F_5}(n, d_{reg}, k) = O \left(q^k \binom{n - k + d_{reg}(k)}{d_{reg}(k)}^\omega \right), \tag{1}$$

where $d_{reg}(k)$ stand for the degree of regularity of the system after fixing the values of k variables.

Determining the degree of regularity for a specific polynomial system is difficult, but for a certain class of systems, called semi-regular systems, it is known that the degree of regularity can be deduced from the number of equations m and the number of variables n [2,8]. In particular, for quadratic semi-regular systems the degree of regularity is the degree of the first term in the power series of

$$S_{m,n}(x) = \frac{(1 - x^2)^m}{(1 - x)^n}$$

with a non-positive coefficient. This gives a practical method to calculate the degree of regularity of any semi-regular system. Empirically, polynomial systems that are randomly chosen have a very large probability of being semi-regular and it is conjectured that most systems are semi-regular systems. For the definition and the theory of semi-regular systems we refer to Chap. 3 of the PhD thesis of Bardet [2].

2.2 Quantum Algorithms

Currently, there are no specialized quantum algorithms that solve polynomial systems over finite fields. However, Grover’s algorithm [13] can be used to speed up the brute force part of the hybrid approach. This approach gives a quadratic speedup for the brute force part of the attack, so the new complexity would be

$$C_{\text{HybridF}_5(n,d_{\text{reg}},k)} = O\left(q^{k/2} \binom{n-k+d_{\text{reg}}(k)}{d_{\text{reg}}(k)}^\omega\right), \quad (2)$$

where the difference with (1) is that we have the factor $q^{k/2}$ instead of q^k . However it should be noted that this approach requires sequentially running $q^{k/2}$ Gröbner basis computations on a quantum computer. This would be an incredible feat because even for moderately sized polynomial systems this would require gigabytes worth of qubits and days of computation without decoherence. Also, note that the gains of parallelizing Grover search grow only with the square root of the number of independent computers used, instead of a linear growth for the classical brute force search [28]. Nevertheless, in the security analysis of the signature scheme proposed in this paper we will be cautious and assume that these kinds of attacks on the MQ problem are possible and we will make our parameter choices accordingly. This has the additional benefit of providing a large safety margin against classical attacks.

Remark 1. Typically the optimal value of k , i.e. the number of variables that is guessed by brute force, is quite small (eg. 2, 3 or 4), this does *not* mean that the hybrid approach is only a marginal improvement over a direct Gröbner basis computation. Guessing only a few variables can drastically reduce the degree of regularity of a system. For example, guessing only one variable in a determined semi-regular system of polynomials roughly reduces the degree of regularity by half! The idea of lifting a public key to an extension field is a countermeasure to the hybrid approach. By working in a large extension field (eg. $\mathbb{F}_{2^{64}}$) we ensure that guessing even a single variable is computationally too expensive.

3 The UOV Signature Scheme

The UOV or Unbalanced Oil and Vinegar digital signature scheme is a multivariate quadratic (MQ) signature scheme. It is a slightly modified version of the original Oil and Vinegar signature scheme that was proposed by Patarin in 1997 [20]. With the right parameter choices UOV has withstood all cryptanalysis since 1997 and it is one of the best studied and most promising MQ signature schemes.

3.1 Description of UOV

The UOV signature scheme uses a one-way function $\mathcal{P} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$, which is a multivariate quadratic polynomial map over some finite field \mathbb{F}_q . The trapdoor

is a factorization $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$, where $\mathcal{T} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^n$ is an invertible linear map, and $\mathcal{F} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$ is a quadratic map whose components f_1, \dots, f_m are of the form

$$f_k(\mathbf{x}) = \sum_{i=1}^v \sum_{j=i}^n \alpha_{i,j,k} x_i x_j + \sum_{i=1}^n \beta_{i,k} x_i + \gamma_k,$$

where $v = n - m$. We say that the first v variables x_1, \dots, x_v are the vinegar variables, whereas the remaining m variables are the oil variables. The components of \mathcal{F} are quadratic polynomials in the variables x_i such that there are no quadratic terms which contain two oil variables. One could say that the vinegar variables and the oil variables are not fully mixed, which is where their names come from.¹

How does the trapdoor $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$ help to invert the function \mathcal{P} ? Given a target $\mathbf{x} \in \mathbb{F}_q^m$ a solution \mathbf{y} for $\mathcal{P}(\mathbf{y}) = \mathbf{x}$ can be found by first solving $\mathcal{F}(\mathbf{y}') = \mathbf{x}$ for \mathbf{y}' and then computing $\mathbf{y} = \mathcal{T}^{-1}(\mathbf{y}')$. The system $\mathcal{F}(\mathbf{y}') = \mathbf{x}$ can be solved efficiently by randomly choosing the values of the vinegar variables. If we substitute these values in the equations the remaining system only contains linear equations, because every quadratic term contains at least one vinegar variable and thus turns into a linear or constant term after substitution. The remaining linear system can be solved using linear algebra. In the event that there are no solutions we can simply try again with a different choice for the vinegar variables.

The trapdoor function is then combined with a collision resistant hash function $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbb{F}_q^m$ into a signature scheme using the standard hash-and-sign paradigm. The resulting key generation, signature generation and verification algorithms of the UOV signature scheme are described in Algorithms 1, 2 and 3.

Algorithm UOVGenerateKeys

input: Random bits to generate \mathcal{F} and \mathcal{T}
output: \mathcal{P} — A public key
 $(\mathcal{F}, \mathcal{T})$ — A corresponding secret key

- 1: $\mathcal{F} \leftarrow$ A randomly chosen UOV system
- 2: $\mathcal{T} \leftarrow$ A randomly chosen linear map $\mathbb{F}_q^n \rightarrow \mathbb{F}_q^n$
- 3: $\mathcal{P} \leftarrow \mathcal{F} \circ \mathcal{T}$
- 4: **return** \mathcal{P} and $(\mathcal{F}, \mathcal{T})$

Algorithm 1. The UOV key pair generation algorithm

¹ However it is not a very good name because in reality oil mixes with oil and vinegar mixes with vinegar but no mixing happens between oil and vinegar, and this is not what happens in UOV polynomials. A better name would have been hen variables and rooster variables because hens can get along with hens and roosters, but two roosters start a fight when they appear in the same term. Moreover, this foreshadows the fact that in order for the signature scheme to be secure, the number of hen (vinegar) variables should be larger than the number of rooster (oil) variables. Nevertheless, we will stick to the traditional naming of oil and vinegar variables.

Algorithm UOVSign

```

input:  $(\mathcal{F}, \mathcal{T})$  — A secret key
          $M$  — A message to sign
output:  $\mathbf{s}$  — A signature for the message  $M$ 
1:  $\mathbf{h} \leftarrow \mathcal{H}(M)$ 
2: while No solution  $\mathbf{s}'$  to the system  $\mathcal{F}(\mathbf{s}') = \mathbf{h}$  is found do
3:   | Assign random values to the first  $v$  entries of  $\mathbf{s}'$ 
4:   | Substitute these values into  $\mathcal{F}(\mathbf{s}') = \mathbf{h}$  to get a linear system  $L(\mathbf{o}) = \mathbf{h}$ .
5:   | if  $L(\mathbf{o}) = \mathbf{h}$  has solutions then
6:     | | Calculate an assignment  $\mathbf{o}$  to the oil variables such that  $L(\mathbf{o}) = \mathbf{h}$ 
7:     | | Assign the entries of  $\mathbf{o}$  to the last  $m$  entries of  $\mathbf{s}'$ 
8:     | end if
9:   end while
10:  $\mathbf{s} \leftarrow \mathcal{T}^{-1}(\mathbf{s}')$ 
11: return  $\mathbf{s}$ 

```

Algorithm 2. The UOV signature generation algorithm

Algorithm UOVVerify

```

input:  $\mathcal{P}$  — A public key
          $M$  — A message
          $\mathbf{s}$  — A candidate signature
output: True if  $\mathbf{s}$  is a valid signature for  $M$ , False otherwise
1:  $\mathbf{h} \leftarrow \mathcal{H}(M)$ 
2:  $\mathbf{h}' \leftarrow \mathcal{P}(\mathbf{s})$ 
3: if  $\mathbf{h} = \mathbf{h}'$  then
4:   | return True
5: else
6:   | return False
7: end if

```

Algorithm 3. The UOV signature verification algorithm

3.2 Attacks Against UOV

Direct Attack. This attack tries to forge a signature s for a message M by solving the polynomial system $\mathcal{P}(s) = \mathcal{H}(M)$. An attacker can use the trick of Thomae and Wolf [27] to reduce this to finding a solution of a polynomial system with $m + 1 - \lfloor n/m \rfloor$ equations. The best known algorithms to solve this problem use the hybrid approach [5] which was briefly described in Sect. 2. Empirically, the systems that have to be solved behave like semi-regular systems [12], therefore we can calculate the degree of regularity and use this to estimate the complexity of the hybrid approach. Petzoldt [22] uses a similar method to estimate the complexity of a direct attack against UOV, the only difference being that we have used an updated estimate of the complexity of F_5 [6]. In Petzoldt's thesis it was shown that the estimated complexity of a direct attack agrees very well

with the measured complexity of a direct attack against small instances of UOV. These experiments justify ignoring the big- O notation in formula (1) and treating the formula as an estimate for the concrete hardness of the hybrid approach.

Example 1. We will estimate the complexity of a direct attack against UOV with the parameter set $(q = 31, m = 52, v = 104)$; this set is proposed in [22] as a set that achieves 128-bit security. Using the trick of Thomae et al. we can reduce finding a solution to this underdetermined system to finding a solution of a determined system with $52 + 1 - \lfloor (52 + 104)/52 \rfloor = 50$ equations. We assume this system to be semi-regular. If we fix k extra variables the degree of regularity is equal to the degree of the first term in the power series of

$$S_{50,50-k}(x) = \frac{(1 - x^2)^{50}}{(1 - x)^{50-k}}$$

which has a non-positive coefficient. For $k = 0$ we have $S_{50,50}(x) = (1 + x)^{50}$, so the degree of regularity is 51. For $k = 1$ we have

$$S_{50,49}(x) = 1 + 49x + 1175x^3 + \dots + 4861946401452x^{25} - 4861946401452x^{26} + O(x^{27}),$$

where all the omitted terms have positive coefficients, so the degree of regularity is 26. We can now use (1) to estimate the complexity of the hybrid approach. We prefer to err on the side of caution, so we have chosen $\omega = 2$ for the value of the linear algebra constant. For k equal to 0 and 1 this is equal to

$$\binom{50 + 51}{51}^2 \approx 2^{194.7} \quad \text{and} \quad 31 \binom{50 - 1 + 26}{26}^2 \approx 2^{137.8}$$

respectively. Continuing this for higher values of k we eventually see that the optimal value of k is 6, the corresponding degree of regularity is 16 and the complexity of the direct attack is $2^{123.9}$.

In the example we concluded that the complexity of the attack is less than 2^{128} which was supposed to be the security level of the parameter set $(q = 31, m = 52, v = 104)$ according to [22]. Even though we have used roughly the same method of estimating the complexity as the method used by Petzoldt [22] we arrive at a slightly different value because we have used a tighter bound on the complexity of F_5 coming from an improved analysis of the hybrid approach [6].

With this method we can calculate the minimal number of equations that is needed in a determined semi-regular system in order to guarantee that the complexity of finding a solution is larger than a targeted security level. For quantum attackers, we can follow the same method with (2) instead of (1) for estimating the complexity of the hybrid approach. The result of these calculations for the security levels of 2^{128} and 2^{256} for different finite fields of size up to $q = 2^{100}$ are plotted in Fig. 1.

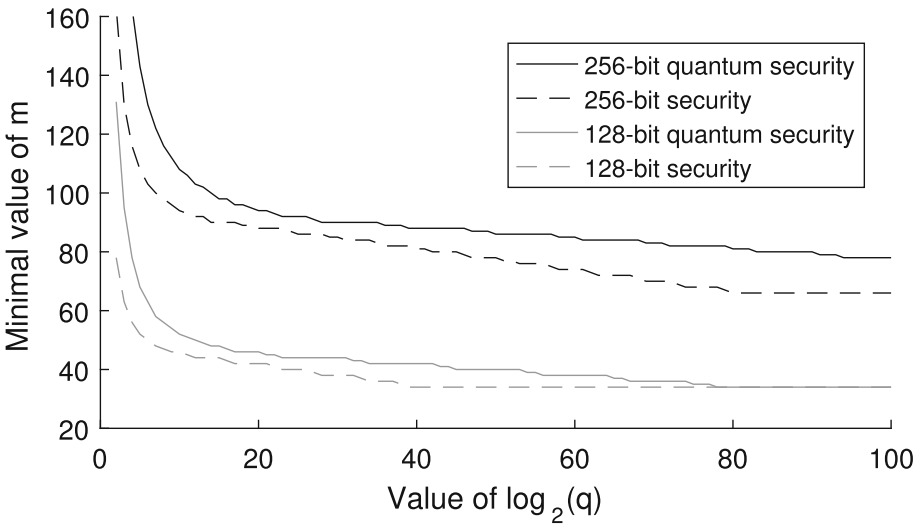


Fig. 1. The minimal sizes of determined semi-regular systems to reach 128-bit security and 256-bit security for different finite fields.

UOV Attack. Patarin [20] suggested in the original version of the Oil and Vinegar scheme to choose the same number of vinegar and oil variables, or $v = m$. This choice was cryptanalyzed by Kipnis and Shamir [16]: they showed that an attacker can find the inverse image of the oil variables under the map \mathcal{T} . This is enough information to find an equivalent secret key, so this breaks the scheme. This approach generalizes for the case $v > m$; the complexity then increases to $O(q^{v-m}n^4)$ [15] and is thus exponential in $v - m$. Typically one chooses $v = 2m$ or $v = 3m$ to preclude the UOV attack.

UOV Reconciliation Attack. Similar to the UOV attack, the UOV reconciliation attack proposed by Ding et al. [9] tries to find an equivalent secret key. We present a brief summary. In this section we will make a distinction between m , the number of polynomials in the public and private system, and o , the number of oil variables. In the UOV signature scheme these numbers are the same, which explains why we did not need to make this distinction before. It turns out that for a public key \mathcal{P} there exists with a very high probability a private key $(\mathcal{F}, \mathcal{T})$ such that the matrix representation of \mathcal{T} is of the form

$$\mathbf{M}_{\mathcal{T}} = \begin{pmatrix} \mathbf{I}_v & \mathbf{T} \\ 0 & \mathbf{I}_o \end{pmatrix}.$$

This means that an attacker only has to find the $v \times o$ matrix \mathbf{T} to get an equivalent key. The UOV reconciliation attack tries to find \mathbf{T} algebraically by solving a quadratic system. If the choice of \mathcal{T} is correct (i.e. there exists a private key of the form $(\mathcal{F}, \mathcal{T})$), then we have that the matrix representation \mathbf{P}_i of the

quadratic part of each polynomial in the public key satisfies for all $1 \leq i \leq m$

$$\begin{pmatrix} *_{v \times v} & *_{o \times v} \\ *_{v \times o} & 0_{o \times o} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_v & 0 \\ -\mathbf{T} & \mathbf{I}_o \end{pmatrix} \mathbf{P}_i \begin{pmatrix} \mathbf{I}_v & -\mathbf{T} \\ 0 & \mathbf{I}_o \end{pmatrix}. \tag{3}$$

The condition that the lower right $o \times o$ submatrices of the private system consist of zeroes give quadratic equations in the entries of \mathbf{T} . It looks like we have o^2 equations for each component, but since the matrix representations are only defined up to the addition of a skew symmetric matrix this gives only $o(o+1)/2$ equations per component. In total we have a system of $mo(o+1)/2$ equations in vo variables. The reconciliation attack tries to recover \mathbf{T} by solving this system of equations.

The reconciliation system has a structure that makes it much easier to solve compared to a random system of the same size. In fact, Ding et al. argue that the complexity of this attack for UOV variants with $v \leq m$ (like Rainbow and TTS) is the same as the complexity of solving a system of m equations in v variables [9].

In the case $v \geq m$ the complexity of the attack is more difficult to estimate, but we can formulate a lower bound to the complexity of the attack. The reconciliation system has $mo(o+1)/2$ equations in ov variables. For all parameter choices of UOV this is a heavily overdetermined system, so it should not be a surprise that there is only one matrix \mathbf{T} that satisfies (3). Computer experiments have shown that there is a unique solution for \mathbf{T} as soon as the number of equations of the reconciliation system exceeds the number of variables. Let $\text{Rec}[v, o, m]$ denote the complexity of a key reconciliation attack against a UOV public system with v vinegar variables, o oil variables and m polynomials in the public key. Increasing m only makes the reconciliation attack easier. Indeed, increasing the number of equations can only make the attack easier, because an attacker could just ignore the extra equations and still find the same unique solution. In other words, if $m < m'$, then we have $\text{Rec}[v, o, m] \geq \text{Rec}[v, o, m']$, provided that $mo(o+1)/2 > ov$, which is the case for all good UOV parameter choices.

We can now derive a lower bound on the complexity of a reconciliation attack when $v > m = o$. According to the above observation, we can increase m , the number of equations, until it matches the number of vinegar variables v , and this would make solving the system easier, i.e. we have

$$\text{Rec}[v, m, m] \geq \text{Rec}[v, m, v]. \tag{4}$$

We can now use the argument of Ding et al. which says that when $m \geq v$, the complexity of the reconciliation attack is equal to the complexity of solving a system of m quadratic equations in v variables, so $\text{Rec}[v, m, v]$ is equal to the complexity of solving a system of v quadratic equations in v variables.

We conclude that a UOV reconciliation attack on a UOV system with m equations and $v \geq m$ vinegar variables is at least as difficult as solving a system of v quadratic variables in v equations, but it is expected to be more difficult,

because a lot of hardness is lost in the inequality (4). In particular, the reconciliation attack is less effective against the UOV scheme than attacking the system $\mathcal{P}(s) = \mathcal{H}(M)$ directly.

Quantum Attacks. There are no known specialized quantum algorithms that solve multivariate quadratic equations. However, as described in Sect. 2, Grover’s algorithms can be used to speed up the exhaustive search part of hybrid solution finding algorithms. This quantum version of the hybrid approach algorithm can be used to speed up a direct attack and a reconciliation attack.

Grover search could be used to speed up the UOV attack from $O(q^{v-m}n^4)$ to $O(q^{\frac{v-m}{2}}n^4)$. This requires repeatedly running an algorithm that calculates the common eigenspaces of a set of matrices and checks whether any of these eigenspaces lies within the oil subspace in superposition. In comparison with the classical algorithm this has the disadvantage that it cannot be parallelized without a significant amount of overhead.

4 Improving UOV

In [22] Petzoldt presented a new method to reduce the public key size of UOV by roughly a factor 8. The key generation algorithm was adapted to make it possible to choose a large part of the public key. One can generate this part with a pseudo-random number generator and replace a large part of the public key by a seed. Also, it is possible to choose part of the public key in such a way such that signatures can be verified faster [21].

Usually, during key generation, a UOV system \mathcal{F} and an invertible linear map \mathcal{T} are chosen randomly, and then \mathcal{P} is determined as $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$. With this strategy we have full control over \mathcal{F} , but no control over the public key \mathcal{P} . Instead, Petzoldt proposed to first pick \mathcal{T} and $v(v+1)/2 + mv$ coefficients of each polynomial of \mathcal{P} . Then we solve the system $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$ to find the coefficients of \mathcal{F} , and the remaining coefficients of \mathcal{P} . This is a linear system of equations, so this can happen efficiently. With a small probability this system does not have any solutions, but in that case we can simply try again with a different choice of \mathcal{T} . For the details of this method we refer to [22].

With this approach the public key size is decreased with $m(v(v+1)/2 + mv)$ field elements, at the negligible cost of including the seed for the random number

Table 1. The effect of Petzoldt’s method on the public key size

security level	q	(m, v)	public key (kB)	public key with Petzoldt’s method (kB)
100-bit	2^8	(36,72)	207	23
128-bit	2^8	(47,94)	460	52
192-bit	2^8	(72,144)	1648	185
256-bit	2^8	(98,196)	4150	464

generator. The public key size is now $m^2(m+1)/2 \log_2(q) + |\text{seed}|$. Table 1 shows that this method drastically reduces the size of the public key. However, the public key remains much larger than the signature schemes that are in use today such as RSA [25] and DSA [17], which typically stay well under 1 kB. Note that if Petzoldt's method is used, the size of the public key is independent of v , the number of vinegar variables.

5 Lifting \mathcal{P} to an Extension Field

In this section we will work with UOV over a finite field \mathbb{F}_{2^r} of characteristic 2. The parameter r is quite important for the security of the scheme, the signature size and key sizes. It can be seen in Fig. 1 that by choosing a larger value of r we can put a smaller number of equations in the system and still reach the same level of security. Since the number of field elements in the public key and secret key is $O(m^3)$ it is desirable to have a small value of m . However, since it costs r bits to store a field element r should not be too big either. We must make a trade-off between large r and large m . In this section we propose a scheme that gets some security benefits of a high value of r , but has a public and private key with coefficients in \mathbb{F}_2 , greatly reducing the key sizes.

5.1 Description of the New Scheme

As usual, the public key of the scheme represents a quadratic system over \mathbb{F}_{2^r} , given by

$$\mathcal{P} = \mathcal{F} \circ \mathcal{T}.$$

When we want to sign a message m we use a hash function to generate a digest of mr bits which represents a vector \mathbf{h} of m elements of \mathbb{F}_{2^r} . Then we use the knowledge of the private key to solve the system $\mathcal{P}(\mathbf{s}) = \mathbf{h}$ to get a valid signature \mathbf{s} . However, the difference with standard UOV is that we now choose all the coefficients of \mathcal{F}, \mathcal{P} and \mathcal{T} in \mathbb{F}_2 . Therefore the key generation process is identical to the key generation process of a regular UOV scheme over \mathbb{F}_2 . In particular, we can use the approach of Petzoldt [22] as explained in Sect. 4 to reduce the size of the public key. Contrary to the key generation, the signature generation and verification still happen over the field \mathbb{F}_{2^r} as usual.

To summarize, we simply take a key pair of the UOV scheme over \mathbb{F}_2 , and use it as a key pair for the UOV scheme over \mathbb{F}_{2^r} . The public key is thus approximately a factor r smaller than if we were to use the regular UOV scheme over \mathbb{F}_{2^r} since we only use one bit to represent each coefficient instead of r bits. Furthermore, we can now choose r to be much larger than what would otherwise its optimal value. This in turn allows for a smaller value of m (See Fig. 1), reducing the public key size even more.

The public key consists of a seed for a pseudorandom number generator and the part of the public map which cannot be generated. The total size of the public key is therefore

$$|\text{seed}| + \frac{m^2(m+1)}{2} \text{ bits.}$$

Storing the private maps \mathcal{F} and \mathcal{T} would take

$$m \frac{v(v+1)}{2} + m^2 v \text{ bits and } n^2 \text{ bits}$$

respectively, but they do not need to be stored, because they can be calculated using the key generation algorithm each time they are needed. A signature consists of $n = m + v$ elements of \mathbb{F}_{2^r} , so the size of the signature is nr bits.

Remark 2. Though we have presented this scheme with a finite field of characteristic 2 and with the subfield $\mathbb{F}_2 \subset \mathbb{F}_{2^r}$, it is easy to see that we can use this scheme with any field extension of finite fields $K \subset K'$. In such a scenario we generate a key pair with coefficients in the small field K , and the signing and verifying is done with elements of the big field K' .

5.2 Security Analysis of the New Scheme

Direct Attack. This attack tries to forge a signature for a certain message M by trying to find a solution $s \in \mathbb{F}_{2^r}^n$ for the system $\mathcal{F}(s) = \mathcal{H}(M)$. The best known methods for this use the hybrid approach as described in Sect. 2.

For a direct attack against the new scheme all the coefficients of the system that needs to be solved lie in \mathbb{F}_2 , except those of the constant terms, because those coefficients come from the message digest. We claim that this does not significantly reduce the hardness of finding solutions relative to the case where the coefficients are generic elements of \mathbb{F}_{2^r} . It has been noticed by Faugère and Perret [12] that the polynomial systems that result from fixing $\approx v$ variables in a UOV system behave like semi-regular systems. The degree of regularity of a quadratic semi-regular system is given by the degree of the first term in the power series of

$$\frac{(1 - x^2)^m}{(1 - x)^n}$$

with a non-positive coefficient. In particular the degree of regularity does not depend on q for semi-regular systems. Hence, the degree of regularity for a direct attack against the modified UOV scheme is identical to the degree of regularity of an attack against the regular UOV scheme. Therefore a Gröbner basis computation against the modified scheme is not significantly more efficient than a Gröbner basis computation against regular UOV with the same parameters. This argument is confirmed by the experimental data in Table 2. There we see that a direct attack is slightly faster against the modified scheme than against the original UOV scheme, but only by a small constant factor. Even though the Gröbner basis is computed over \mathbb{F}_{2^r} , the largest part of the arithmetic only involves the field elements 0 and 1, so the arithmetic is faster than with generic elements of \mathbb{F}_{2^r} . This is where the difference observed in Table 2 comes from. If we do the same experiment with a smaller extension field such as \mathbb{F}_{2^8} there is no observed difference between the running time of a direct attack against a regular UOV scheme and our modified scheme.

Table 2. Running time of a direct attack against the regular UOV scheme over $\mathbb{F}_{2^{64}}$ and the modified UOV scheme, with the MAGMA v2.22-10 implementation of the F4 algorithm. We did not implement the method of Thomae and Wolf [27].

(m,v)	Regular UOV (s)	Lifted UOV (s)	difference
(7,35)	0.43	0.21	-52%
(8,40)	1.56	0.76	-51%
(9,45)	7.00	3.21	-54%
(10,50)	33.50	17.44	-48%
(11,55)	132.88	76.60	-42%
(12,60)	828.31	588.33	-29%

Remark 3. In a direct attack one fixes $\approx v$ variables randomly to make the system a slightly overdetermined system. In our experiments we have fixed these variables to values in \mathbb{F}_2 to make sure that we do not introduce linear terms with coefficients in \mathbb{F}_{2^r} instead of \mathbb{F}_2 in the case of the modified UOV scheme.

Remark 4. It might seem tempting to decompose the equations over \mathbb{F}_{2^r} into equations over \mathbb{F}_2 to make a direct attack more efficient. This decomposition is done by fixing some basis β_1, \dots, β_r of \mathbb{F}_{2^r} over \mathbb{F}_2 and replacing each variable x_i by $\sum_{j=1}^r \hat{x}_{i,j} \beta_j$, where the $\hat{x}_{i,j}$ are nr new variables in \mathbb{F}_2 . Each equation of the original system is then decomposed into r equations, resulting in a total of mr equations in nm variables over \mathbb{F}_2 . The problem with this approach is that the number of equations and variables is increased by the factor r , which makes the naive approach of solving the decomposed system with a generic boolean solver hopelessly slow. However, the decomposed system has a specific structure which could potentially be exploited to solve the system more efficiently. We investigated this possibility, but we were not able to make any progress. It should be pointed out that this idea does not only apply to our scheme, but to any multivariate cryptosystem over a field of non-prime order. Still, no such attacks are reported in literature. One could say that the idea of decomposing a system to make it easier to solve is not very promising because in big-field schemes such as Gui [24] and medium-field schemes such as HMFev [23] the systems are decomposed with the objective of making them *harder* to solve for an attacker.

Key Recovery Attacks. In contrast to a direct attack, the modified scheme is more vulnerable to a key recovery attack. Since the key pair used in the Lifted UOV scheme is identical to the key pair of regular UOV over the field \mathbb{F}_2 it is clear that a key recovery attack against the Lifted UOV scheme is equivalent to a key recovery attack against a regular UOV scheme over \mathbb{F}_2 , which is much easier than a key recovery attack against UOV over \mathbb{F}_{2^r} . Luckily, key recovery attacks against UOV have been investigated ever since the invention of the oil and vinegar scheme in 1997 [20], so it is well understood which attacks are

possible (see Sect. 3.2) and what the complexities of these attacks are. It is also clear that we can make key recovery attacks harder by increasing the number of vinegar variables.

The UOV attack attempts to recover an equivalent private key by searching for the oil subspace. This attack has complexity $q^{v-m-1} \cdot n^4$. Since a UOV attack on the Lifted UOV scheme is equivalent to a UOV attack over \mathbb{F}_2 , we have that the complexity of a UOV attack against the Lifted UOV scheme is $2^{v-m-1} \cdot n^4$.

The reconciliation attack against the lifted UOV scheme is equivalent to the UOV reconciliation attack against UOV over the field \mathbb{F}_2 . A lower bound on the complexity of this attack is given by the complexity of solving a quadratic system of v variables and v equations over \mathbb{F}_2 , but we expect the problem to be harder. There exists specialized algorithms for solving polynomial systems over \mathbb{F}_2 that are more efficient than the generic hybrid approach. One method is a smart exhaustive search, which requires approximately $\log_2(n)2^{n+2}$ bit operations [7]. The BooleanSolve algorithm [3] combines an exhaustive search with sparse linear algebra to achieve a complexity of $O(2^{0.792n})$. However the method only becomes faster than the exhaustive search method when $n > 200$. Recently, Joux et al. proposed a new algorithm that was able to solve a boolean system of 146 quadratic equations in 73 variables in one day [14]. The algorithm beats the exhaustive search algorithm, even for small systems. The complexity of this algorithm is still under investigation, but a rough estimate based on the reported experiments suggests that it scales like $2^{\alpha n}$ with α between 0.8 and 0.85 and with a small constant factor. For choosing the parameters of our signature scheme, we have assumed that a determined system of n quadratic boolean equations provides $0.75n$ bits of security, even though this is likely to seriously overestimate the capabilities of the state of the art algorithms. Quantum attackers can use Grover search to solve systems over \mathbb{F}_2 with complexity $O(2^{n/2})$.

5.3 Choice of Parameters

For convenience and efficiency we will work with binary finite fields whose elements are represented by a number of bits that is a multiple of 16, i.e. the finite fields we want to use are $\mathbb{F}_{2^{16}}, \mathbb{F}_{2^{32}}, \mathbb{F}_{2^{48}}$ and so on.

When designing a signature scheme of security level l , we choose a finite field that is large enough such that the minimal number of equations in a determined regular system that is needed to reach the security level l is minimized. Figure 1 shows that for 128-bit and 256-bit security the chosen fields are $\mathbb{F}_{2^{48}}$ and $\mathbb{F}_{2^{80}}$ respectively, and the minimal number of equations is 34 and 66 respectively or 40 and 81 when considering quantum attacks. For 100-bit and 192-bit security the chosen fields are $\mathbb{F}_{2^{32}}$ and $\mathbb{F}_{2^{64}}$, and the minimal number of equations is 27 and 50 for classical attackers or 33 and 60 for quantum attackers.

We now consider the constraints on the parameters due to the different attacks against our scheme. In order to be safe against a direct attack we require

$$m - \lfloor v/m \rfloor \geq m_{min},$$

Table 3. Parameter choices and corresponding public key and signature sizes for different security levels

Security level		(r, m, v)	pk (kB)	sig (kB)	classical security
100-bit	classical	(32,31,134)	1.9	0.6	
	quantum	(32,37,200)	3.2	0.9	115 bit
128-bit	classical	(48,38,171)	3.4	1.2	
	quantum	(48,45,256)	5.7	1.8	153 bit
192-bit	classical	(64,54,256)	9.8	2.4	
	quantum	(64,65,384)	17.0	3.5	224 bit
256-bit	classical	(80,70,341)	21.2	4.0	
	quantum	(80,87,526)	40.7	6.0	296 bit

with m_{min} equal to 27, 34, 50 or 66 if the desired security level is 100 bits, 128 bits, 192 bits, or 256 bits respectively. For quantum attackers m_{min} is equal to 33, 40, 60 and 81 respectively. In order to be safe against the UOV attack we require

$$2^{v-m-1}n^4 > 2^l \quad \text{or} \quad 2^{(v-m-1)/2}n^4 > 2^l,$$

depending on whether we want l bits of security against classical, or quantum adversaries. To be secure against the UOV reconciliation attack it suffices that an attacker cannot solve a determined system with v equations over \mathbb{F}_2 . Therefore it suffices to have

$$2^{0.75v} > 2^l \quad \text{or} \quad 2^{v/2} > 2^l$$

for classical and quantum attackers respectively. The parameter sets displayed in Table 3 satisfy all the constraints for the targeted security level and minimize the size of the public key, i.e. they minimize m . In the last column of the table, the bit complexity of the best known classical attack against the parameter set is calculated. For all the proposed parameters the best known classical attack is a direct Groebner basis attack.

5.4 Trade-Off

In comparison to regular UOV, Lifted UOV has much smaller public keys, but also larger signatures. In the discussion above, we have chosen the parameter r very large in order to minimize the size of the public key, without considering the size of the signatures. It is possible to make a trade-off between the size of the public key and the size of the signature by choosing a smaller value of r . Having a smaller value of r requires a larger value of m to reach the same security level, resulting in a larger public key, but since the signature consists of n elements of \mathbb{F}_{2^r} it also leads to smaller signatures. Figure 2 compares public key sizes and signature sizes of the Lifted UOV scheme with different values of the parameter r with some other MQ signature schemes [22], the lattice-based signature scheme BLISS-II [10] and SPHINCS, a hash-based signature scheme [4]. Note that even

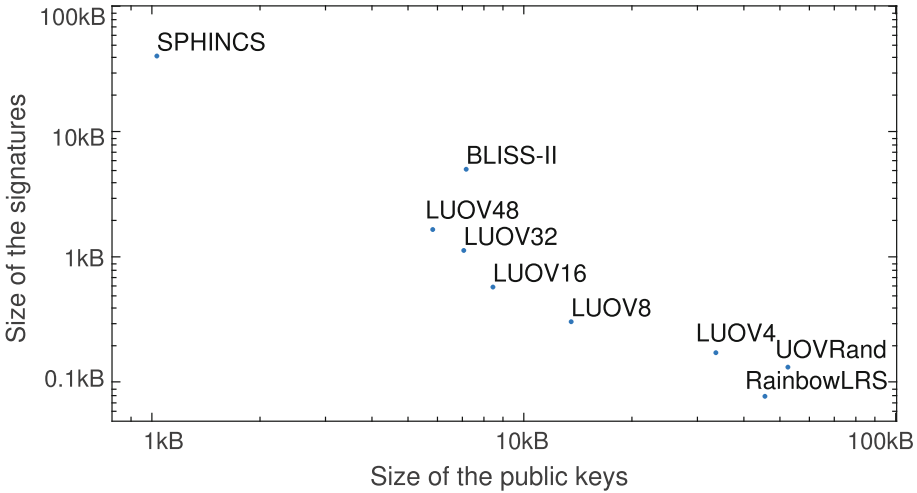


Fig. 2. Comparison of different signature schemes providing 128 bits of post-quantum security.

though the MQ schemes UOVRand and RainbowLRS2 claim to provide 128-bit of post-quantum security, their parameters are not chosen to resist quantum attacks on the MQ problem or quantum versions of the UOV attack. So we are not comparing schemes with the same security level. Ignoring quantum attacks, the Lifted UOV signature scheme with $r = 48$ in the comparison achieves 153 bits of security.

Example 2. For some application on a low-cost device it might be desirable to have a signature scheme that provides 128 bits of post-quantum security with minimal signature sizes subject to the condition that the public key is smaller than, say, 10 kB. If we choose the parameters as in the discussion above, we would have a public key of 5.7 kB and signatures of 1.8 kB. However, we can do better by choosing $r = 12$. The lowest values of m and v providing 128 bits of security are then $m = 54$ and $v = 256$. This leads to a public key of 9.8 kB (< 10 kB) and a signature of 0.45 kB.

6 Implementation and Results

We developed an ANSI C implementation of the Lifted UOV signature scheme. The large fields are implemented as extension fields of $\mathbb{F}_{2^{16}}$ and the arithmetic in $\mathbb{F}_{2^{16}}$ is done using log tables. We have a table that maps each nonzero element x to the number y such that $x = a^y$, where a is some generator of the group $\mathbb{F}_{2^{16}}^\times$. Conversely, we also have a table that maps a number y to the element a^y . Multiplication in $\mathbb{F}_{2^{16}}$ is then computed with three table lookups and an addition modulo $2^{16} - 1$. Note that this approach could make our implementation vulnerable to cache timing attacks. Newer CPUs support the CLMUL instruction set

Table 4. Running times for the key generation, signing and verification algorithms on a single thread on an Intel[®]Core[™] i7-4710MQ CPU at 2.5 GHz

Security level		key gen (ms)	sig gen (ms)	verification (ms)
100-bit	classical	4	6	3
	quantum	13	16	7
128-bit	classical	10	14	7
	quantum	26	34	15
192-bit	classical	32	46	21
	quantum	148	156	54
256-bit	classical	125	149	55
	quantum	366	410	144

which could be used to perform the field arithmetic efficiently without the need for lookup tables, eliminating the possibility of this attack. Two field elements are added using a XOR operation. During the key generation phase we only use elements of \mathbb{F}_2 , so we have used bit slicing whenever possible to speed up the algorithm. The running times of the key generation, signature generation and the verification algorithms are displayed in Table 4.

Please note that the implementation uses naive implementations of matrix multiplication, polynomial multiplication and Gaussian reduction, and the code was not heavily optimized. Therefore, it can be expected that the running times reported in Table 4 are nowhere near optimal. Some techniques that can speed up the code very significantly include writing cache friendly code, using parallelization and using Karatsuba’s algorithms for the field arithmetic. Moreover, it is possible to use a method of Petzoldt to structure part of the public key in such a way that the verification algorithm is faster [22]. In order to avoid storing the large private key, part of the key generation algorithm is run each time a signature is generated to generate the private key. If a batch of messages is signed together this step only has to happen once. Alternatively, if storing the private key is not an issue, this part can be omitted altogether to speed up the signing algorithm significantly.

7 Conclusion

The simple idea of lifting a UOV key pair from \mathbb{F}_2 to an extension field \mathbb{F}_{2^r} increases the security against direct attacks without affecting the size of the public key. At the same time, thanks to the method of Petzoldt, we can increase the number of vinegar variables to protect against key recovery attacks without increasing the size of the public key. These two ideas come together to create a secure signature scheme whose public key is an order of magnitude smaller than other MQ signature schemes, with slightly larger signatures. The signature scheme is very competitive with other post-quantum signature schemes.

By choosing the parameter r it is possible to make a trade-off between larger public keys and smaller signatures or vice versa. We developed a rudimentary ANSI C implementation of the Lifted UOV signature scheme which shows that key generation, signing and verification takes only a few milliseconds for 100-bit security instantiations of the scheme and up to a few hundred milliseconds for 256-bit security instantiations. However it is very likely that these times can be improved significantly with an optimized implementation.

The idea of lifting keys to a large extension field can be applied to any MQ signature scheme, but it might not always be useful to produce smaller public keys. We believe that the idea could be used to improve the Rainbow signature scheme, but not HFE or C^* . This is because the public keys of signature schemes such as HFE and C^* are not semi-regular maps [11] and have a much smaller degree of regularity than random maps of the same dimensions. This means that guessing a few variables does not necessarily reduce the degree of regularity, like it does in the case of semi-regular systems. This makes the hybrid approach unsuitable for attacking these systems, since solving the system with one big Gröbner basis computation is more efficient. Therefore there is no point in lifting the system to a larger field, because the complexity of a Gröbner basis computation is largely independent of the size of the finite field.

Acknowledgements. This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported by the European Commission through the Horizon 2020 research and innovation programme under grant agreement No. H2020-ICT-2014-644371 WITDOM and H2020-ICT-2014-645622 PQCRYPTO.

References

1. PQCRYPTO ICT-645622 (2015). <http://pqcrypto.eu.org/>
2. Bardet, M.: Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie. Ph.D. thesis, Université Pierre et Marie Curie-Paris VI (2004)
3. Bardet, M., Faugère, J.C., Salvy, B., Spaenlehauer, P.J.: On the complexity of solving quadratic Boolean systems. *J. Complex.* **29**(1), 53–75 (2013)
4. 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
5. Bettale, L., Faugère, J.C., Perret, L.: Hybrid approach for solving multivariate systems over finite fields. *J. Math. Cryptol.* **3**(3), 177–197 (2009)
6. Bettale, L., Faugère, J.C., Perret, L.: Solving polynomial systems over finite fields: improved analysis of the hybrid approach. In: Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation, pp. 67–74. ACM (2012)
7. Bouillaguet, C., Chen, H.-C., Cheng, C.-M., Chou, T., Niederhagen, R., Shamir, A., Yang, B.-Y.: Fast exhaustive search for polynomial systems in \mathbb{F}_2 . In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 203–218. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_14

8. Diem, C.: The XL-algorithm and a conjecture from commutative algebra. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 323–337. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30539-2_23
9. Ding, J., Yang, B.-Y., Chen, C.-H.O., Chen, M.-S., Cheng, C.-M.: New differential-algebraic attacks and reparametrization of rainbow. In: Bellovin, S.M., Gennaro, R., Keromytis, A., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 242–257. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68914-0_15
10. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_3
11. Faugère, J.-C., Joux, A.: Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Gröbner bases. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_3
12. Faugère, J.C., Perret, L.: On the security of UOV. IACR Cryptology ePrint Archive 2009, 483 (2009)
13. 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. ACM (1996)
14. Joux, A., Vitse, V.: A crossbred algorithm for solving Boolean polynomial systems. IACR Cryptology ePrint Archive 2017, 372 (2017)
15. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 206–222. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_15
16. Kipnis, A., Shamir, A.: Cryptanalysis of the oil and vinegar signature scheme. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 257–266. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0055733>
17. Kravitz, D.W.: Digital signature algorithm, US Patent 5,231,668, 27 July 1993
18. Michael, R.G., David, S.J.: Computers and Intractability: A Guide to the Theory of NP-Completeness. WH Free. Co., San Francisco (1979)
19. National Institute for Standards and Technology (NIST): Post-quantum crypto standardization (2016). <http://csrc.nist.gov/groups/ST/post-quantum-crypto/>
20. Patarin, J.: The oil and vinegar signature scheme. In: 1997 Dagstuhl Workshop on Cryptography (1997)
21. Petzoldt, A.: Hybrid approach for the fast verification for improved versions of the UOV and Rainbow signature schemes. IACR Cryptology ePrint Archive 2013, 315 (2013)
22. Petzoldt, A.: Selecting and Reducing Key Sizes for Multivariate Cryptography. Ph.D. thesis, TU Darmstadt, referenten: Professor Dr. Johannes Buchmann, Professor Jintai Ding, Ph.D, July 2013
23. Petzoldt, A., Chen, M.-S., Ding, J., Yang, B.-Y.: HMFev - an efficient multivariate signature scheme. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 205–223. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_12
24. Petzoldt, A., Chen, M.-S., Yang, B.-Y., Tao, C., Ding, J.: Design principles for HFEv- based multivariate signature schemes. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 311–334. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_14
25. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)

26. Shor, P.W.: Polynomial time algorithms for discrete logarithms and factoring on a quantum computer. In: Adleman, L.M., Huang, M.-D. (eds.) ANTS 1994. LNCS, vol. 877, p. 289. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58691-1_68
27. Thomae, E., Wolf, C.: Solving underdetermined systems of multivariate quadratic equations revisited. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 156–171. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_10
28. Zalka, C.: Grover's quantum searching algorithm is optimal. *Phys. Rev. A* **60**(4), 2746 (1999)

Gabidulin Matrix Codes and Their Application to Small Ciphertext Size Cryptosystems

Thierry P. Berger^(✉), Philippe Gaborit, and Olivier Ruatta

Univ. Limoges, CNRS, XLIM, UMR 7252, 87000 Limoges, France
thierry.berger@unilim.fr

Abstract. In this paper we propose a new method to hide the structure of Gabidulin codes for cryptographic applications. At the difference of previous cryptosystems based on Gabidulin codes, we do not try to mask the structure of Gabidulin codes by the use of some distortion methods, but we consider matrix codes obtained from subcodes of binary images of Gabidulin codes. This allows us to remove the properties related to multiplication in the extension field. In particular, this prevents the use of Frobenius for cryptanalysis. Thus, Overbeck's attack can no longer be applied. In practice we obtain public key with a gain of a factor of order ten compared to the classical Goppa-McEliece scheme with still a small cipher text of order only 1 kbits, better than recent cryptosystems for which the cipher text size is of order 10 kbits. Several results used and proved in the paper are of independent interest: results on structural properties of Gabidulin matrix codes and hardness of deciding whether a code is equivalent to a subcode of a matrix code.

Keywords: McEliece public key cryptosystem · Rank metric
Gabidulin codes

1 Introduction

McEliece introduced cryptography based on error-correcting codes in 1978. There are two main concerns regarding this type of cryptosystems, namely the size of the public keys and the fact that the system is not reduced to a regular well known hard problem such as the Syndrome Decoding problem for random codes. Over the years several approaches were attempted to decrease the size of the public keys. Two main approaches give potential interesting results allowing to potentially decrease the size of the key from megabits to kilobits using compact public keys like in [5, 21, 22] using a group action on the public matrix or using another metric as the rank metric [13]. Using the rank metric was the first approach proposed in 1991. The gain comes from the fact that attacking the generic decoding problem for rank metric is practically harder (in term of best known attacks) than for Hamming metric [16], which leads to smaller key size for the same security.

Previous Work. There is a large literature on McEliece like public key cryptosystems based on rank metric and Gabidulin codes (a rank metric analog of

Reed-Solomon codes) [13, 18, 26]. Most of the times, the designers of such cryptosystems try to mask the structure using distortion techniques. Unfortunately, there exists a cryptanalysis method essentially based on the action of the Frobenius map on Gabidulin code, the so-called Overbeck attack. This attack uses the very special structure of Gabidulin codes to distinguish modified Gabidulin codes from random codes. Overall this attack led to the cryptanalysis of most of cryptosystem based on Gabidulin codes (see [24, 25] and its generalizations [23]). The core of this attack is a distinguisher on Gabidulin codes which is directly derived from the application of the Frobenius map on this code. All the cryptosystems using Gabidulin codes are defined over an extension field and the Frobenius map can be applied to the public code. Notice that there is an equivalent type of distinguishing attack on Reed-Solomon codes, the so-called “square attack” [9].

The Overbeck’s distinguisher attack is the main obstacle to the use of Gabidulin codes for a cryptographic purpose in a McEliece type setting. Recently a new approach was proposed by Loidreau [18], mixing Gabidulin codes with LRPC codes [15]. This approach seems to sufficiently hide the Gabidulin structure. This leads to public keys with sizes not as low as the original GPT system, where a factor 100 could be gained compared to the classical McEliece using Goppa codes, but to a smaller by an order of 10.

Besides the McEliece approach, the RQC system based on decoding random quasi-cyclic codes in rank metric was also proposed in [1]. This approach, based on Alekhovich approach [2], has the advantage to rely on decoding random (quasi-cyclic) codes without hidden structure, but the decoding efficiency of the considered LRPC codes is in $O(\sqrt{n})$ for cryptographic purposes, when it is in $O(n)$ for Gabidulin codes, so that these systems lead to rather small public keys but with cipher of approximatively the same size (of order 5–10 Kbits) of the public key. The same type of phenomenon appears for Hamming metric for MDPC and HQC cryptosystems [1, 22].

Hence using Gabidulin codes has the potential to reach very small cipher text (as for McEliece) of only a few thousand bits. To sum up, the McEliece classical approach leads to cryptosystem where the public key is very large but the cipher can be very small. The approaches using quasi-cyclic lead to the same small parameters (but not *very* small) for both the public key AND the cipher. Although all these quasi-cyclic approaches are of clear interest in terms of general security (there is almost no hidden structure (MDPC) or no hidden structure at all (HQC)), it is of independent interest to obtain cryptosystems with very small size of ciphers (of order 1 Kbits) (comparable to classical McEliece and smaller than quasi-cyclic based approaches) but at the cost of a larger public key.

Our Contribution and Main Idea. In this paper we propose a new approach to hide the structure of the Gabidulin code in order to resist to Overbeck’s distinguishing attack. We obtain parameters with public key size gain of order 10 compared to classical McEliece scheme but with very small ciphers (1 Kbits) comparable to McEliece (with Goppa codes) but better by a factor of 5 to 10 compared to Loidreau recent approach (for similar public key size).

Our main idea to prevent the use of the Frobenius map and Overbeck’s attack, is to consider Gabidulin codes not as linear codes over the large extension field (the way they are defined by Gabidulin) but as linear over the base field $GF(q)$ (the way Gabidulin codes were originally described by Delsarte [10]), the code is then considered as a matrix code over the base field. This approach has two consequences: firstly, it breaks the linearity over the large field leads to a larger way to describe the code in itself (and hence a larger key size), secondly, it makes the structure of the Gabidulin code harder to recover, especially when not all the matrix code is given but only a subcode of the Gabidulin matrix code.

The approach, hence permits to hide the Gabidulin structure (at the cost of losing a factor on the size of the key) but preserves the very good decoding properties of Gabidulin codes. The security analysis of our system relies on the indistinguishability of equivalent Gabidulin matrix subcodes with random matrix codes.

Paper Organization. The paper is organized as follows: Sect. 2 gives the necessary background on matrix codes and Gabidulin codes, Sect. 3 describes our new protocol based on subcodes of Gabidulin matrix codes, Sect. 4 considers the security of the protocol, and at last Sect. 5 gives practical instantiations for our protocol.

2 Background on Matrix Codes, Gabidulin Codes and Gabidulin Matrix Codes

We will use the following notations:

- $GF(q)$ is the finite field with q elements. For most applications, $q = 2$, however, in the last section $q = 2^u$ for a small u .
- $GF(q^m)$ is an extension of $GF(q)$ of degree m .
- $GL(m, q)$ is the General Linear Group of the $m \times m$ invertible matrices with entries in $GF(q)$.

2.1 Matrix Codes

In this section, we will recall some classical results on matrix codes in rank metric, q -ary image of a code over $GF(q)^m$ and Gabidulin codes. Section 2.2 contains also a new efficient method that allows to rebuild a code over $GF(q^m)$ from one of its q -ary image.

Definition 1. *An $m \times n$ -matrix code C is a subspace of the $GF(q)$ -vector space of $m \times n$ matrices over $GF(q)$.*

Let $E = GF(q)^m$ be the set of m -tuple over $GF(q)$.

Definition 2. [4] *A q -ary E -code of length n is a $GF(q)$ subspace of E^n .*

Clearly, an E -code of dimension k can be considered as an $[nm, k]$ $GF(q)$ -linear code. Moreover, there is a natural bijection between matrix codes and E -codes, which consists in writing the coefficients of an E -code in column.

Such a code can be also considered as a $GF(q)$ -linear code of length n over $GF(q^m)$. We fix a basis $\mathcal{B} = (b_1, \dots, b_m)$ of $GF(q^m)$ over $GF(q)$ and denote by $\phi_{\mathcal{B}}$ the corresponding $GF(q)$ -linear isomorphism $GF(q^m) \mapsto GF(q)^m$. Applying $\phi_{\mathcal{B}}^{-1}$ to each coordinate of an E -codeword, or equivalently to each column of an $m \times n$ matrix over $GF(q)$, the codewords can be considered as elements of $GF(q^m)^n$.

Note that the image of an matrix code by $\phi_{\mathcal{B}}^{-1}$ is a $GF(q)$ -linear code over $GF(q^m)$, but a priori, it is not a $GF(q^m)$ -linear code since such codes has no special properties with respect to product by elements of $GF(q^m)$.

In the sequel, we will use indifferently the representation of codewords as $m \times n$ matrices or as elements of E^n . The first one is best suited to properties related to the rank of elements, the second one allows to use the background techniques of Coding Theory, in particular those that are related to generator matrices.

Rank Distance of Matrix Codes. As explained in [4], the advantage of an E -code is its ability to correct burst errors. Consequently, we want to look at its block-error correcting capacity.

For instance, the Hamming weight of the matrix representation of a codeword is the number of its non-zero columns. In the sequel, we are not interested by the Hamming metric, but by the rank metric. We give the definitions in the context of the matrix representation of codewords, because it is the most natural for the notion of rank metric.

Definition 3. Let $c \in E^n$ be a codeword, and M_c the corresponding $m \times n$ matrix with entries in $GF(q)$. The rank weight of c is the rank of M_c .

Definition 4. The minimum distance of an matrix code C is the minimum of the rank-weight of its non-zero elements.

Remark that determining a codeword of minimum distance in such a code is exactly solving the min-rank problem for the corresponding matrix code.

An important question is those of isometries for the rank distance. Let $B_0 \in GL(q, m)$ and $B_1 \in GL(q, n)$ be two invertible matrices with entries in $GF(q)$ of respective sizes m and n . If c is a codeword, then $rank(B_0 M_c B_1) = rank(M_c)$. In consequence, we can define the notion of equivalent matrix codes.

If B_0 and B_1 are defined as previously, we denote by Ψ_{B_0, B_1} the map $c \mapsto c'$ such that $M_{c'} = B_0 M_c B_1$.

Definition 5. Two E -codes C and C' are equivalent for the rank metric if there exists a map Ψ_{B_0, B_1} such that $C' = \Psi_{B_0, B_1}(C)$.

The following proposition describes the action of the rank-distance equivalence on the E -code representation of matrix codes.

Proposition 1. *Let $C' = \Psi_{B_0, B_1}(C)$. If G is an $km \times nm$ generator matrix of C , then $G' = G \times (B_0^T \otimes B_1)$ is a generator matrix of C' .*

Proof. This is a direct consequence of the fact that the action $Mc \mapsto B_0Mc$ is a multiplication of each column of Mc by B_0 which is transposed in the E -code representation as a multiplication of each m -tuple of E by the transposed matrix B_0^T on the right.

2.2 q -Ary Image of a $GF(q^m)$ -Linear Code

From a $GF(q^m)$ -linear code, it is possible to derive an E -code using the map $\phi_{\mathcal{B}}$ defined in Sect. 2.1.

Definition 6. *Let $\mathcal{B} = (b_1, \dots, b_m)$ be a basis of $GF(q^m)$ over $GF(q)$ and denote by $\phi_{\mathcal{B}}$ the corresponding isomorphism $GF(q^m) \mapsto GF(q)^m$. If $\Phi_{\mathcal{B}}$ is the action of $\phi_{\mathcal{B}}$ on each coordinates of codewords in $GF(q^m)^n$, the q -ary image $Im_q(\mathcal{C})$ relative to \mathcal{B} of a $GF(q^m)$ -linear code of length n is the image of \mathcal{C} by $\Phi_{\mathcal{B}}$.*

Notice that the definition of q -ary image depends on the choice of the basis \mathcal{B} . The main difference between \mathcal{C} and $Im_q(\mathcal{C})$ is the fact that, if \mathcal{B} is not known, $Im_q(\mathcal{C})$ is no more $GF(q^m)$ -linear but only $GF(q)$ -linear. If the dimension of \mathcal{C} is k , those of $Im_q(\mathcal{C})$ is km .

Even when the basis \mathcal{B} is known, for practical implementation in the context of error correcting codes, $q = 2$ and the more efficient way to encode binary messages with a code over $GF(2^m)$ is to use a generator matrix of $Im_q(\mathcal{C})$. As a consequence, the size of this $GF(q)$ generator matrix is m times greater than those of the original code over $GF(q^m)$.

There exists an efficient way to construct a $GF(q)$ -generator matrix G of $Im_q(\mathcal{C})$ from a $GF(q^m)$ -generator matrix \mathcal{G} of \mathcal{C} . Let α be a primitive root of $GF(q^m)^*$. The map $\sigma_{\alpha} : x \mapsto \alpha x$ is a $GF(q)$ -linear automorphism of $GF(q^m)$. The image of σ_{α} by the map $\phi_{\mathcal{B}}$ is a linear isomorphism of $E = GF(q)^m$. We denote by $M_{\alpha} \in GL(q, m)$ the corresponding matrix. Note that M_{α} depends on the choice of \mathcal{B} . For instance, if we choose for the basis \mathcal{B} the multiplicative basis $(1, \alpha, \dots, \alpha^{m-1})$, then M_{α} is nothing else than the companion matrix of the minimal polynomial of α .

For an element $\beta = \alpha^i \in GF(q^m)^*$, the corresponding matrix is $M_{\beta} = M_{\alpha}^i$. In addition, M_0 is the $m \times m$ zero matrix.

Proposition 2. *If $\mathcal{G} = (\beta_{i,j})$ is an $k \times n$ generator matrix of \mathcal{C} , then the block matrix $G = (M_{\beta_{i,j}})$ is an $km \times nm$ generator matrix of $Im_q(\mathcal{C})$.*

The following proposition describes the link between the duality over $GF(q)$ and the duality over $GF(q^m)$.

Proposition 3. *Let \mathcal{C} be a code over $GF(q^m)$ and \mathcal{C}^{\perp} be its $GF(q^m)$ -dual. The dual of the q -ary image of \mathcal{C} relative to the basis \mathcal{B} is the q -ary image of its dual \mathcal{C}^{\perp} relative to the dual basis \mathcal{B}^* of \mathcal{B} .*

Proof. Let $\mathcal{G} = (\beta_{i,j})$ and $\mathcal{H} = (\gamma_{i,j})$ be respectively be a generator matrix of \mathcal{C} and its $GF(q^m)$ dual \mathcal{C}^\perp . This implies $\mathcal{G} \times \mathcal{H}^T = 0$.

Using the injection $\beta \in GF(q^m) \mapsto M_\beta \in GL_{q^m} \cup \{M_0\}$, we obtain the relation

$$M_{\beta_{i,j}} \times (M_{\gamma_{i,j}}^T)^T = 0.$$

The set of matrices $\{M_\beta^T \mid \beta \in GF(q^m)\}$ is another matrix representation of the finite field $GF(q^m)$ which corresponds to the choice of the dual basis of \mathcal{B} .

Reconstruction of a q^m -Linear Code from Its q -Ary Image. Given a $GF(q)$ -generator matrix of the q -ary image of a code, a natural problem is to recover the $GF(q^m)$ structure even if the basis \mathcal{B} is not known.

Let \mathcal{C} be the original linear code over $GF(q^m)$. Up to a permutation of its coordinates, we can suppose that \mathcal{C} admits a generator matrix under systematic form $\mathcal{G} = (\mathcal{I}_k \mid \mathcal{R})$, $\mathcal{R} = (\beta_{i,j})_{1 \leq i \leq k, 1 \leq j \leq n-k}$. A first consequence is that, for all its q -ary image, up to a permutation of its m -tuple coordinates (which corresponds to a permutation of the columns in matrix representation), this code admits a systematic generator matrix $G = (I_{km} \mid R_{km,(n-k)m})$.

We suppose that we know G and we want to recover \mathcal{B} and \mathcal{C} (or an equivalent representation).

The q -ary image of \mathcal{G} relative to \mathcal{B} is the block matrix $G = (I_{km} \mid R)$ with $R = (M_{\beta_{i,j}})$. Since the systematic generator matrix of a code is unique, from one can recover the matrices $M_{\beta_{i,j}}$. All the non-zero matrices $M_{\beta_{i,j}}$ represents an element of the same cyclic group of order $q^m - 1$.

Suppose that there is at least one matrix M_β of order $q^m - 1$, then we can construct a representation of $GF(q^m) = GF(q)(\beta)$, where β is a root of the minimal polynomial of M_β and we can identify each coefficient $M_{\beta_{i,j}} = M_\beta^v$ to β^v . This leads to a $GF(q^m)$ generator matrix \mathcal{G}' which is not necessary \mathcal{G} but such that G can be considered as a q -ary image of \mathcal{G}' .

If we do not found a generator matrix of the cyclic group of order $q^m - 1$, there are only two possibilities: if all the matrices are in a cyclic group of order $q^{m'} - 1$ for a divisor m' of m , the code \mathcal{C} is a code over a subfield $GF(q^{m'})$ of $GF(q^m)$ with an extension of scalars from $GF(q^{m'})$ of $GF(q^m)$. If this condition is not verified, at least one matrix has a minimal polynomial of degree m and can be used by linear combination to reconstruct the full finite field representation.

2.3 Gabidulin Codes

In [12], Gabidulin introduced the notion of rank metric. In this article he gave a bound analogous to that of Singleton in the context of Hamming metric. For a code \mathcal{C} of length $n \leq m$ over the finite field $GF(q^m)$, the rank bound is equal to the Singleton bound, *i.e.* $k + d = n + 1$, where d is the minimum rank distance of \mathcal{C} . In addition, he presented a family of MRD (Maximum Rank Distance) which meet this bound. These codes are known as Gabidulin codes.

Definition 7. Let $\mathcal{G} = (g_1, g_2, \dots, g_n)$ be an ordered set of $n \leq m$ elements of $GF(q^m)$, which are linearly independent over $GF(q)$. The Gabidulin code $Gab_{\mathcal{G},k}$ of support \mathcal{G} and dimension k is the code generated by the generator matrix

$$Gab_{\mathcal{G},k} = \begin{pmatrix} g_1^{[0]} & g_2^{[0]} & \cdots & g_n^{[0]} \\ g_1^{[1]} & g_2^{[1]} & \cdots & g_n^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ g_1^{[k-1]} & g_2^{[k-1]} & \cdots & g_n^{[k-1]} \end{pmatrix} \quad \text{with the convention } g_j^{[i]} = g_j^{q^i}.$$

These codes can be viewed as evaluation of linearized polynomials of linear degree strictly less than k over a set of linearly independent points. Using a similar reasoning than for Reed Solomon codes, it is easy to show that these codes are MRD (and so are MDS). There is an algorithm in polynomial times which corrects any error of rank weight up to $\lfloor (n - k)/2 \rfloor$.

A first remark is the fact that the support \mathcal{G} of a Gabidulin code is defined up to a scalar multiplication of \mathcal{G} . So, it is always possible to fix $g_1 = 1$.

In addition, it is easy to recover the support (i.e. \mathcal{G}) of a Gabidulin from a generator matrix of this code.

Let θ_q be the Frobenius map acting on $GF(q^m)$ and Θ_q its extension on codewords of $GF(q^m)^n$. One can notice that the inverse of θ_q is the map $\theta_{q^{m-1}} : \beta \mapsto \beta^{q^{m-1}}$. Let $\Theta_{q^{m-1}}(Gab_{\mathcal{G},k})$ be the image of $Gab_{\mathcal{G},k}$ by $\Theta_{q^{m-1}}$. We have the following property $\Theta_{q^{m-1}}(Gab_{\mathcal{G},k}) \cap Gab_{\mathcal{G},k} = Gab_{\mathcal{G},k-1}$. By iterating this process $k - 1$ times, we obtain $Gab_{\mathcal{G},1}$ and any non-zero element of $Gab_{\mathcal{G},1}$ gives \mathcal{G} up to a scalar multiplication.

There has been a lot of attempt to use these codes and rank metric in cryptography [7, 13, 14, 18]. Unfortunately, due to the high $GF(q^m)$ linearity of Gabidulin codes, most of them, except [18], were cryptanalyzed.

Our aim is to use the q -ary images of Gabidulin codes to destroy the $GF(q^m)$ underlying structure. The main payload of our approach is the fact that it increases the public key of a factor m , since it is no more possible to describe our codes over $GF(q^m)$.

Maximum-Rank Array Codes. In [27] Roth introduced a notion of maximum-rank array codes, which is nothing else than q -ary images of Gabidulin codes. As the notion of matrix code is more general, we will use the following definition.

Definition 8. A Gabidulin matrix code (a GM code) is a $m \times n$ matrix code which is equivalent to the q -ary image of a Gabidulin code in the meaning of Definition 5.

A first remark is the fact that, if the parameters of construction of a GM code are known, the decoding algorithm of Gabidulin codes holds for GM codes.

In addition, applying results of Sects. 2.2 and 2.3, it is easy to recover the structure of an underlying Gabidulin code and the basis \mathcal{B} from a generator matrix of a GM code.

2.4 From GM Codes to Gabidulin Codes

A natural problem related to GM codes is the following:

Problem 1. Given a generator matrix of a GM code C , recover a Gabidulin code $Gab_{\mathcal{G},k}$, a basis \mathcal{B} of $GF(q^m)$ over $GF(q)$, a matrix $B_0 \in GL(q, m)$ and a matrix $B_1 \in GL(q, n)$ such that C is equivalent by Ψ_{B_0, B_1} to the q -ary image of $Gab_{\mathcal{G},k}$ relative to the basis \mathcal{B} .

The problem of equivalence of Gabidulin codes has been studied in [3]. In particular, it is easy to see that the matrix $B_1 \in GL(q, n)$ in Definition 5 corresponds to a change of support $\mathcal{G}' = \mathcal{G}B_1$ of the Gabidulin code. So the choice of B_1 can be interpreted as the choice of a basis of the $GF(q)$ -vector space generated by \mathcal{G} .

Without loss of generality, one can reformulate the previous problem as follows:

Problem 2. Given a generator matrix of a GM code C , recover a Gabidulin code $Gab_{\mathcal{G},k}$, a basis \mathcal{B} of $GF(q^m)$ over $GF(q)$ and a matrix $B_0 \in GL(q, m)$ such that C is equivalent by Ψ_{B_0, I_n} to the q -ary image of $Gab_{\mathcal{G},k}$ relative to the basis \mathcal{B} .

Let $G = (M_{\beta_{i,j}})$ be the generator matrix of $Im_q(Gab_{\mathcal{G},k})$ as defined in Proposition 2.

The matrix $G' = G \times (B_0 \otimes I_n) = (M_{\beta_{i,j}} B_0)$ is a generator matrix of the GM code. We deduce that the matrix $(B_0^{-1} \otimes I_k) \times G' = (B_0^{-1} M_{\beta_{i,j}} B_0)$ is a generator matrix of the GM code. For $\beta \in GF(q^m)$ we define $M'_\beta = B_0^{-1} M_\beta B_0$. The set $\{M'_\beta \mid \beta \in GF(q^m)\}$ is another matrix representation of the finite field $GF(q^m)$ corresponding to a change of basis associated to B_0 .

We have proved the following theorem:

Theorem 1. *Let C be the GM code defined by a Gabidulin code $Gab_{\mathcal{G},k}$, a basis \mathcal{B} and a map Ψ_{B_0, B_1} . Let $V_{\mathcal{G}}$ be the $GF(q)$ -vector space of dimension n generated by \mathcal{G} . There is a basis \mathcal{G}' of $V_{\mathcal{G}}$ and a basis \mathcal{B}' of $GF(q^m)$ such that C is the q -ary image of the Gabidulin code $Gab_{\mathcal{G}',k}$ relative to the basis \mathcal{B}' .*

The problem is now the following: given a generator matrix G of a GM code of dimension km , recover \mathcal{G} and \mathcal{B} such that C is the q -ary image relative to \mathcal{B} of the Gabidulin code $Gab_{\mathcal{G},k}$.

It can be done in two steps:

1. Recovering the $GF(q^m)$ -structure of the q -ary image. This can be done using the method described in Sect. 2.2.
2. Reconstruction of the parameters of a Gabidulin code from a $GF(q^m)$ -generator matrix of this code. This second problem is solved in Sect. 2.3.

3 A New Cryptosystem Based on Equivalent Subcodes of Gabidulin Matrix Codes

This section presents our new proposition of McEliece like cryptosystem. We saw in the previous section that recovering the structure of a Gabidulin code directly from matrix code structure is easy, hence in order to hide this structure it is necessary to break the regular structure of the Gabidulin matrix code by considering a subcode of it. The core idea of our proposal is then to use subcodes of q -ary images of Gabidulin in order to destroy the multiplicative structure on the extension field and to stop classical attacks such as Overbeck.

Our proposition is a McEliece like cryptosystem which uses $GF(q)$ -subcodes of GM codes. By taking a $GF(q)$ -subcode of a matrix code, we destroy the underlying $GF(q^m)$ structure inherited from the original $GF(q^m)$ -linear code.

In order to facilitate the implementation and to allow a simpler enumeration, we limit ourself to codes C which admit a systematic generator matrix, and subcodes of q -ary codes which also admit a systematic generator matrix without additional permutation. This restriction does not decrease the security of our protocol. Indeed, up to a permutation of coordinates any code admits a systematic generator matrix.

If $G = (I_{km} | R)$ is a $GF(q)$ generator matrix of a q -ary image code of parameters $[nm, km]$, then $H = (-R^T | I_{(n-k)m})$ is a generator matrix of the dual. Let s be an integer which corresponds to the targeted loss of dimension. In order to construct any subcode of C , we randomly choose a $s \times km - s$ matrix L with coefficients in $GF(q)$. Set $U = (L | I_s | 0_{s, (n-k)m})$. The matrix U is a $s \times n$ matrix of rank s . We construct a parity matrix $H' = \begin{pmatrix} U \\ H \end{pmatrix}$.

In the sequel, we fix $n = m$, which leads to Gabidulin codes having the greatest possible length. In practical application, $q = 2$ or $q = 2^u$ for small u (typically $1 \leq u \leq 5$).

Algorithm 1. Derivation Key Algorithm

- Choose a random matrix $B_0 \in GL(q, n)$.
 - Choose a random matrix $B_1 \in GL(q, n)$.
 - Compute $G' = \Psi_{B_0, B_1}(G)$. Let $G_{\text{sys}} = (I_{kn} | R)$ be the systematic generator matrix of the code C' generated by G' . Set $H = (B^T | I_{(n-k)n})$.
 - Choose a random $s \times kn - s$ q -ary matrix L . Set $U = (L | I_s | 0_{s, (n-k)n})$. Set $H' = \begin{pmatrix} U \\ H \end{pmatrix}$.
 - Public key: the systematic generator matrix G_{pub} of the code C_{pub} with parity check matrix H' .
 - Secret keys: B_0 and B_1
-

Note that the matrix L is not secret, since it can be computed from the inverse systematic form of the dual of C_{pub} . However, its knowledge does not allow to recover H .

The code C_{pub} is an $n \times n$ matrix code of q -ary dimension $k' = km - s = (k - 1)n$. The knowledge of B_0 and B_1 (and indeed those of \mathcal{G} and \mathcal{B}) allows to decode up to t rank errors on C_{pub} .

Algorithm 2. Encryption

- Let $x \in GF(2)^{km-s}$ be the message to encrypt.
 - Using G_{pub} , compute the corresponding matrix codeword c .
 - Choose randomly a $n \times n$ matrix e of rank t .
 - Output the cipher text $y = c + e$.
-

Algorithm 3. Decryption

- Using the secret parameters and the Gabidulin decoder, recover e from y .
 - Recover x from $c = y + e$.
-

4 Security Analysis

In this section, we consider the security of our problem. First we consider the general problem of subcode equivalence problem for rank metric that is proven hard, then we define our security assumption that the Gabidulin matrix subcodes are indistinguishable from random matrix codes. There are essentially two families of attacks: the structural attacks, where the attacker tries to find the secret key or an equivalent secret key, and the decoding brute force attacks, which use decoding techniques for random codes. Under our security assumption the best brute force attacks correspond to attacking the MinRank problem (the Rank Syndrome problem in the case of matrix codes with rank metric), and we give security arguments for our assumption regarding recovering the structure of the Gabidulin equivalent subcode.

4.1 Subcode Equivalence of Matrix Codes Is NP-complete

We are interested in the following problem: given two linear codes C and D , is there a linear isometry for rank metric f such that $f(D)$ is a subcode of C ?

We will first have a look on this problem in the context of Hamming metric. Let K be a finite field. A monomial transformation is a permutation of coordinates followed by a scalar multiplication of each coordinate by a non-zero element of K . If $K = GF(2)$, a monomial transformation is simply a permutation of coordinates of codewords. It is well-known (see e.g. [17]) that the isometries of K -linear codes for the Hamming metric are exactly the monomial transformations.

The subcode Hamming metric equivalence code for binary codes is as follows:

Problem 3 (Subcode equivalence of binary codes, SEBC). Given two binary codes C and D of parameters $[n; k]$ and $[n; k']$, $k' \leq k$, is there exists an isometry for the Hamming metric π such that $\pi(D)$ is a subcode of C ?

Recently, it was proved in [6] that for binary codes and Hamming metric, this problem is NP-complete, even if checking if a code is a subcode of another is easy, and the equivalence problem itself is easy in most of cases.

The corresponding decision problem for binary array codes and rank metric is the following:

Problem 4 (Subcode equivalence of matrix codes, SEMC). Given two binary $n \times n$ matrix codes C and D . Is there exist an isometry Ψ_{B_0, B_1} such that $\Psi_{B_0, B_1}(D)$ is a subcode of C ?

In this section, we will prove that this problem is also NP-complete.

A binary code C of length n can be embedded in a $n \times n$ matrix code by identifying codewords of length n with diagonal matrices of size $n \times n$. We denote by $Diag(C)$ the corresponding rank code. The most important fact is that the Hamming weight of a codeword is the rank of the corresponding diagonal matrix.

Before proving our result we will recall the MacWilliams extension theorem. The original proof of this theorem is in her thesis [19], another reference is [8].

Theorem 2. *Let C and C' be two linear codes of length n over a finite field K . If there exists a linear isometry for the Hamming distance f such that $f(C) = C'$, this isometry can be extended to the whole space K^n . In other words there exists a monomial transformation ξ of K^n such that f is the restriction of ξ to C .*

Note that this extension theorem is for the Hamming metric. To our knowledge, it was not studied in the context of rank metric.

Theorem 3. *Subcode rank metric equivalence of binary matrix codes is NP-complete.*

Proof. It is sufficient to reduce this problem to the binary codes for the Hamming distance. Suppose that we are able to solve SEMC. Let C and D be the entries of the subcode equivalence problem for binary codes and Hamming distance SEBC. We apply SEMC to their diagonal $n \times n$ matrix representations $Diag(D)$ and $Diag(C)$.

Suppose that there exists an isometry Ψ_{B_0, B_1} which solve this problem. Let $\Delta' = \Psi_{B_0, B_1}(Diag(D))$ be the image of $Diag(D)$ by Ψ_{B_0, B_1} . Since Δ' is a subcode of $Diag(C)$, its elements are diagonal matrices and there exists a binary code D' such that $Diag(D') = \Delta'$. Clearly, D' is a subcode of C . The most important point of the demonstration is the fact that the rank metric isometry Ψ_{B_0, B_1} between $Diag(D)$ and $Diag(D')$ induces a Hamming metric isometry f between D and D' . Applying the MacWilliams extension theorem to f , it can be extended to a permutation of $GF(2)^n$ and the answer to the problem is “yes”.

Reciprocally, if there exists a permutation π of $GF(2)^n$ which solves the problem for the binary codes, the rank metric isometry $\Psi_{\Pi^{-1}, \Pi}$ is a solution of the SEMC problem for the corresponding diagonal matrix codes.

4.2 Security Assumption

Our security assumption can be written as follows:

Assumption 1. *It is hard to distinguish an array code C which is equivalent for the rank metric (in the meaning of Definition 5) to a Gabidulin matrix subcode from a random matrix code with the same length and dimension.*

We saw in the previous section that the general matrix subcode equivalence problem was hard for rank metric, the previous assumption permits to consider the special subcase of equivalent Gabidulin matrix subcodes. In practice the equivalent Gabidulin matrix subcode structure corresponds to the public key of our cryptosystem. In the following we consider several attacks on the equivalent Gabidulin matrix subcode structure to justify our assumption. The next two sections consider the case of the Overbeck’s distinguishing attacks, we then consider algebraic and classical MinRank attacks.

4.3 Reconstruction of a Subcode of a q -Ary Image

Even if the subcode equivalence problem is NP-complete for matrix codes, we have more information since our starting point is the q -ary image. In this context, we can define the following problem:

Let C be a subcode over $GF(q)$ of a q -ary image of a code \mathcal{C} of parameters $[n; k]_{q^m}$ relative to a basis \mathcal{B} . The dimension of C is $k' = km - s$, thus s is the loss of dimension of the subcode. In addition, without loss of generality, we suppose that both \mathcal{C} and C admit a generator matrix under systematic form.

Problem 5 (Reconstruction of a subcode of a q -ary image). Given the systematic generator matrix $G = (I_{k'} \mid R_{k', nm-k'})$ in the E -code representation of C , the integers n, m, k and s , recover \mathcal{C} and \mathcal{B} , or equivalent parameters such that C is a subcode of the q -ary image of \mathcal{C} relative to the basis \mathcal{B} .

We propose two algorithms to solve this problem.

Enumeration of Basis \mathcal{B} . A first idea is to enumerate all the $GF(q^m)$ linear codes with required parameters, their q -ary images, and then to test if the public code is a subcode of this q -ary image. One can notice that, in order to construct all the q -ary images of a code, it is necessary to try almost all possible basis \mathcal{B} .

In fact, it is easy to avoid the enumeration of $GF(q^m)$ linear codes. Suppose we have found the right base \mathcal{B} . Using the inverse of $\Phi_{\mathcal{B}}$ (cf. Definition 6), from any codeword of C one obtains a codeword of \mathcal{C} . If we compute all the codewords associated to the rows of a generator matrix of C , we obtain $km - s$ codewords of \mathcal{C} , which generate by $GF(q^m)$ linearity the code \mathcal{C} , or at least a $GF(q^m)$ subcode of \mathcal{C} . Following this remark, we propose an algorithm to solve our problem. The code C is defined by a systematic generator matrix $G = (I_{k'} \mid R)$.

For a practical implementation, one can compute $\Phi_{\mathcal{B}}^{-1}$ on each row of G and perform simultaneously the Gaussian elimination. If \mathcal{B} is not a good candidate, the test needs in average to test the rank of $k + l$ rows for a small l (typically,

Algorithm 4. Enumeration of basis

- Input: m, n, k, s and $G = (I_{k'} | R)$.
 - For all \mathcal{B} , compute $G' = \Phi_{\mathcal{B}}^{-1}(G)$, *i.e.* apply $\Phi_{\mathcal{B}}^{-1}$ to each row of G .
 - If $\text{rank}_{GF(q^m)}(G') \leq k$ then return \mathcal{B} and G' (a generator matrix of \mathcal{C}).
-

$l = 1$ or 2). In addition, from the fact that G is under systematic form, there is no additional Gaussian elimination for $k + l \leq k'$, but just a normalization of the leading coefficient of each row.

Without going into the technical details, the fact that the rank of the matrix does not increase in a generic way can be observed as soon as we apply Φ^{-1} to the second row of G' .

From a given code \mathcal{C} , a fixed q -ary image is obtained by a basis \mathcal{B} up to a scalar multiplication in $GF(q^m)$, *i.e.* if α is a multiplicative generator of $GF(q^m)^*$, any basis $M_{\alpha}^i \mathcal{B}$, $0 \leq i \leq q^m - 1$, leads to the same q -ary image.

In addition, we have to look at the action of Frobenius map on $GF(q^m)$ -linear codes. We denote by M_q the matrix representation of θ_q relative to the basis \mathcal{B} . It is easy to verify that the q -ary image of \mathcal{C} relative to $M_q \mathcal{B}$ is equal to the q -ary image of $\theta_q(\mathcal{C})$ relative to the basis \mathcal{B} .

We can now evaluate the cost of our algorithm.

Let $N_{q,m} = \prod_{i=0}^{m-1} (q^m - q^i)$ be the size of the linear group $GL(q, m)$.

The full cost of this algorithm is $cN_{q,m}/(m(q^m - 1))$ where c is twice the cost of an inversion and $nm - k'$ multiplications in $GF(q^m)$: $c \approx 2(nm - k')m \times (q - 1)/q$ where $(q - 1)/q$ is the expectation for a coefficient to be nonzero. In addition, $N_{q,m} \approx \text{const } q^{m(m+1)/2}$ with $(q - 1)/q \leq \text{const} \leq 1$.

One can notice that the value of s has practically no effect on this complexity.

For our practical implementation, $m = n = s$, $k \simeq m/2$ and $q = 2$, thus our complexity is approximatively $m^2/4 \times 2^{m(m-1)/2}$.

Reconstruction of the Special Form of the Parity Check Matrix. From G we deduce an anti-systematic (*i.e.* a systematic matrix for the inverse order) parity check matrix $H'' = (R^T | I_{nm-k'})$. A first remark is the fact that the s first rows of matrix H'' gives directly the matrix U in the subcode construction given in Sect. 3. So the matrix L cannot be secret.

To solve our problem, we naturally try to recover the generator matrix of the dual of the form $H' = \begin{pmatrix} U \\ H \end{pmatrix}$.

In a first step we will focus on recovering only one block of m rows of H corresponding to the $GF(q^m)$ structure. Looking at the m last rows of H'' , we know that, for each of these m rows, there exist a linear combination of the rows of L which corresponds to the difference between the m last rows of H and of H'' .

We deduce the following algorithm.

Algorithm 5. Reconstruction of parity check matrix

- Input: m, n, k, s and $H'' = (R^T \mid I_{nm-k'})$.
 - Let L and M be respectively the matrix corresponding to the s first rows, respectively the m last rows, of H' . Set V the code generated by L . For all element $(\ell_1, \dots, \ell_m) \in V^m$, compute $M' = M - (\ell_1, \dots, \ell_m)^T$.
 - Extract the first m columns of M' . Set M_β be the corresponding $m \times m$ matrix. If $M_\beta^{q^m-1} = I_m$, then we have found with high probability a suitable matrix representation of $GF(q^m)$ and we can construct a corresponding basis \mathcal{B}
 - return \mathcal{B} and the corresponding element $(\ell_1, \dots, \ell_m) \in V^m$.
-

We do not specify all details and possible improvements in order to recover completely the code \mathcal{C} . Clearly, the cost of this algorithm is q^{ms} tests. We neglect some marginal operations and consider only the computation of $M_\beta^{q^m-1}$ for each test, which leads to a cost of m^2 products of matrices of size m . The full cost of this algorithm is about $m^4 q^{ms}$ operations.

In our implementation, we choose $s = m$, thus, for this value of s , this attack is more expensive than the previous one.

In conclusion of this paragraph, following Sect. 4.1, we assume that Problem 5 is difficult.

4.4 Reconstruction of Subcodes of GM Codes

Clearly, the methods described in Sect. 4.3 can be directly applied to subcodes of GM codes. The parameters for cryptographic applications must take in account this kind of attack.

The additional problem for subcodes of GM codes is the possibility to use some specific property of Gabidulin codes. The specific properties of Gabidulin codes are derived from the use of the Frobenius map in their construction. It is not surprising that the only known distinguisher against Gabidulin codes is those of Overbeck, which is based on the use of the Frobenius map [24].

Overbeck’s Distinguisher for Gabidulin Codes

- For a random code \mathcal{C} of length n and dimension k , the dimension of the code generated by $\mathcal{C} \cup \Theta_q(\mathcal{C})$ is close to $\min(2k, n)$
- For a Gabidulin code, this dimension is $k + 1$.

Note that this is the dual property we used in Sect. 2.3 in order to recover the support of a Gabidulin code from a generator matrix of the code.

Our problem now is to identify the action of the Frobenius map on a subcode of a GM code. A first remark is that if we identify the action of the Frobenius map on our code, the Overbeck distinguisher holds for subcodes of a GM codes, since the rank of $\mathcal{C} \cup \Theta_q(\mathcal{C})$ is overbounded by $(k+1)m$ instead of $2k' = 2km - 2s$ in the general case.

The Frobenius map is a $GF(q)$ -linear isomorphism of $GF(q^m)$. It admits a matrix representation in $GL(m, q)$. In addition, if the basis \mathcal{B} is a normal basis, this matrix is those of the circular shift. We denote by $S \in GL(m, q)$ the matrix of this circular permutation.

To identify the Frobenius map in a random basis \mathcal{B} , we have to test the action of the conjugates of S by the elements of $GL(m, q)$, and then to apply the Overbeck distinguisher.

Algorithm 6. Overbeck distinguisher on subcodes of GM codes

- Input: m, n, k, s and C given by a generator matrix $G = (I_{k'} \mid R)$.
 - For all $B \in GL(m, q)$, compute $B' = B^{-1}SB$.
 - Compute a generator matrix G' of $C + \Psi_{B, I_n}(C)$.
 - If $rank(G') \leq km$, then return B (which gives the underlying $GL(q^m)$ structure).
-

As previously, a good candidate B is obtained up to the action of a multiple of a Frobenius map and the action of the cyclic matrix group of order $q^m - 1$ corresponding to the scalar multiplications in $GF(q^m)$. Note that each iteration requires a matrix inversion and two matrix multiplications.

The cost of this algorithm is $3m^2N_{q,m}/(m(q^m - 1))$ which is a bit greater than those of Algorithm 4.

In consequence, for $s \geq m/2$, the structural security can be underbounded by the cost of Algorithm 4.

4.5 Algebraic Attacks

We consider the equations obtained from the multiplication of G' by H' . Here, the algebraic attack consists in solving a bihomogeneous system of bidegree $(1, 1)$. We directly apply results from [11] in order to evaluate the complexity of the approach and to test security parameters with respect to this attack. The system has two sets of variables, namely the entries of the matrix B_0 and the entries of the matrix B_1 respectively and the system is homogeneous and of degree at most 1 with respect to these sets of variables. The matrix B_0 has size $m \times m$ and it has m^2 entries and B_1 has size $n \times n$ and it has n^2 entries. Applying bound given in [11], the regularity d_{reg} of the ideal associated to the system is bounded by $\min(m^2 + 1, n^2 + 1)$ ([11] Theorem 6). Then the complexity of solving is bounded by $\mathcal{O}\left(\binom{m^2+n^2+\min(n^2+1, m^2+1)}{\min(n^2+1, m^2+1)}^\omega\right)$ where ω is the exponent of the linear algebra. This is an upper bound of the cost of solving the system, but we implement in magma some tests and the growth of the regularity of the system is closed to the bound even if you had to stop to $m = 8$ because computations did not finished for higher values.

4.6 Decoding Random Matrix Codes for Rank Metric

Decoding a random $m \times n$ matrix code of dimension k' corresponds to solve the problem MinRank on the matrix code to which the encrypted word has been added. The MinRank problem is a well-known NP-hard problem. The cost of the best algorithms to solve this problem is $c \times q^{tk'/n}$, where t is the rank of the error and $c \simeq n^{2.3}$ is the cost of the linear algebra [16].

For our parameters, $n = m$, $s = m$, $q = 2$, $k' = m(k-1)$ and $t = \lfloor (n - k)/2 \rfloor$, which gives $m^{2.3}2^{t(k-1)}$.

5 Practical Applications

In this section, we will first derive directly from Sect. 4 some parameters suitable for cryptographic applications in McEliece like public key encryption schemes.

We also present a variant using an intermediate subfield in order to decrease significantly the size of the public key.

5.1 Parameters for Binary Subcodes of GM Codes

We choose the following parameters: $q = 2$, $n = m$, $k = m/2$ and $s = m$. We fix a basis \mathcal{G} of $GF(2^m)$ over $GF(2)$. We construct a generator matrix of $Gab_{\mathcal{G},k}$.

The minimum rank distance of this code is $d = n - k + 1$, its error correcting capacity is $t = \lfloor (d - 1)/2 \rfloor$.

We choose a basis \mathcal{B} of $GF(2^n)$ over $GF(2)$ and compute a generator matrix G of the q -ary image C of $Gab_{\mathcal{G},k}$ relative to \mathcal{B} .

The matrix G can be public, since the choices of \mathcal{G} and \mathcal{B} are randomized in the derivation key algorithm.

The size of the public key under systematic form is $k'(nm - k') = (km - s)((m - k)m$.

Practical Instantiation. We propose two sets of instantiation, with a respective level of security greater than 128 bits and 256 bits.

The cost of the structural attack is those of the basis enumeration, which is $m^2/4 \times 2^{m(m-1)/2}$. The cost of the brute decoding attack is those of MinRank, which is $m^{2.3}2^{t(k-1)}$.

m	k	Message	Cipher text	MinRank	Structural	Public key
32	16	480 bits	1024 bits	2^{131}	2^{504}	31.8 KB
46	24	1058 bits	2116 bits	2^{265}	2^{1044}	136.6 KB

Table 1. Security: 128 bits

u	m	k	Message	Cipher text	MinRank	Structural	Public key
1	32	16	480 bits	1024 bits	2^{131}	2^{504}	31.8 KB
2	34	18	544 bits	1156 bits	2^{139}	2^{279}	20.3 KB
3	36	16	468 bits	1296 bits	2^{141}	2^{204}	15.7 KB
4	36	16	432 bits	1296 bits	2^{131}	2^{150}	11.3 KB
5	40	15	400 bits	1600 bits	2^{132}	2^{145}	11.7 KB

5.2 Intermediate Subfield

The payload for masking the $GF(2^m)$ structure of a code is the fact that it becomes a $GF(2)$ -linear code. In consequence, it implies a multiplicative factor m on the size of the public key.

A possible compromise to decrease this factor is the following: the starting point remains a Gabidulin code over $GF(2^m)$, however, the q -ary image can be taken on $q = 2^u$, where u is a small divisor of m , typically $2 \leq u \leq 6$. It allows to describe the public code as a linear code over $GF(q)$, which decreases the size of the key of a factor u . Obviously, all security settings must be adjusted to this context.

The most important fact is a change in the decoding algorithms for random codes. Indeed, the matrix codewords have entries in $GF(2^u)$ while the rank these matrices is a binary rank. Clearly, the attacks derived from the MinRank problem remains effective, however, we have also to look at the attacks on decoding $GF(q)$ -linear code for binary rank metric. In practice, for our choice of parameters, decoding attacks are no more effective than solving the corresponding binary MinRank problem.

For the Overbeck attack, the action of the Frobenius must be found by applying the Frobenius $x \mapsto x^2$ to the coordinates of matrix codewords (*i.e.* to elements of $GF(2^u)$), end then by searching a matrix representation of $x \mapsto x^{2^u}$ in $GL_{2^u}(v)$.

So the number of possibilities becomes $N_{v,2^u}/v(2^m - 1) = v^{-1} \prod_{i=1}^{v-1} (q^m - q^i)$ with $q = 2^u$. Note that we do not take in account the cost of a verification. So this security level is not a tight lower bound. In addition, we checked that the action of the Frobenius map without a complementary search of the matrix representation of the second frobenius map does not allow a distinguisher similar to the Overbeck one.

The following tables show some possible parameters having a security level respectively greater than 128 bits and 256 bits. As previously, we fix $m = n$ and $s = m$. We have incorporated our previous proposition into these tables (Tables 1 and 2).

The cost for MinRank remains unchanged: $m^{2.3}2^{t(k-1)}$. k' . The cost of the structural attack becomes $(v(q - 1)/q)^2 \times 2^{m(v-1)/2}$.

Table 2. Security: 256 bits

u	m	k	Message	Cipher text	MinRank	Structural	Public key
1	46	24	1058 bits	2116 bits	2^{265}	2^{1044}	136.6 KB
2	48	23	1008 bits	2304 bits	2^{264}	2^{560}	79.7 KB
3	48	24	1008 bits	2304 bits	2^{264}	2^{367}	53.1 KB
4	52	21	884 bits	2704 bits	2^{268}	2^{319}	49.0 KB
5	55	21	880 bits	3025 bits	2^{285}	2^{281}	46.0 KB

5.3 Comparison with Other Proposals

For the same level of security (at least 128 bits), the original McEliece cryptosystem must use a binary Goppa code of parameters [4096,3556,91] which leads to a public key of size 938 KB. The size of the message must be 3556 bits and the size of the ciphertext 4096 bits.

The only non-broken McEliece like cryptosystem based on rank metric is those presented in [18]. The size of public key are similar to our results. For instance the first proposition of parameters with a security level greater than 128 has the following values: $n = 64$, $m = 96$, $k = 40$, decoding security $\simeq 2^{139}$, key recovery security: $\simeq 2^{188}$, size of public key: 11.5 KB.

The main advantages of our proposition is a smaller length of cipher text (1296 bits versus 6144) and a more efficient decoding algorithm, since we work in the finite field $GF(2^m)$ with $m \leq 40$ instead of $m = 96$ (Table 3).

Table 3. Comparison with other McEliece like PKC, security: 128 bits

McEliece like PKC	Cipher text	Public key
Our proposal	1296 its	11.3 KB
Hidden Gabidulin codes [18]	6144 bits	11.5 KB
Classical Goppa Codes [20]	4096 bits	938 KB

6 Conclusion

In this paper we described a new approach to use the good properties of Gabidulin codes in a McEliece encryption setting. The results we obtain is a good tradeoff for a cryptosystem with a small ciphertext size. We succeed in obtaining a ciphertext of the same order than the classical Goppa-McEliece scheme but with a gain of a factor of order 10 on the size of the public key. Our system is the code-based cryptosystem with the smaller public key size for a cipher text of size of order only 1 Kbit. The paper also presents results of independent interests for code-based cryptography on structural properties of matrix Gabidulin codes or on the hardness of the subcode equivalence matrix codes problem.

References

1. Aguilar Melchor, C., Blazy, O., Deneuville, J.-C., Gaborit, P., Zémor, G.: Efficient encryption from random quasi-cyclic codes. CoRR abs/1612.05572 (2016). <http://arxiv.org/abs/1612.05572>
2. Alekhnovich, M.: More on average case vs approximation complexity. In: Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS 2003), Cambridge, MA, USA, 11–14 October 2003, pp. 298–307 (2003)
3. Berger, T.P.: Isometries for rank distance and permutation group of Gabidulin codes. *IEEE Trans. Inf. Theory* **49**(11), 3016–3019 (2003)
4. Berger, T.P., El Amrani, N.: Codes over $\mathcal{L}(GF(2)^m, GF(2)^m)$, MDS diffusion matrices and cryptographic applications. In: El Hajji, S., Nitaj, A., Carlet, C., Souidi, E.M. (eds.) C2SI 2015. LNCS, vol. 9084, pp. 197–214. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18681-8_16
5. Berger, T.P., Cayrel, P.-L., Gaborit, P., Otmani, A.: Reducing key length of the McEliece cryptosystem. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 77–97. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02384-2_6
6. Berger, T.P., Gueye, C.T., Klanti, J.B.: A NP-complete problem in coding theory with application to code based cryptography. In: El Hajji, S., Nitaj, A., Souidi, E.M. (eds.) C2SI 2017. LNCS, vol. 10194, pp. 230–237. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55589-8_15
7. Berger, T.P., Loidreau, P.: Designing an efficient and secure public-key cryptosystem based on reducible rank codes. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 218–229. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30556-9_18
8. Bogart, K.P., Goldberg, D., Gordon, J.: An elementary proof of the MacWilliams theorem on equivalence of codes. *Inf. Control* **37**, 19–22 (1978)
9. Couvreur, A., Gaborit, P., Gauthier-Umaña, V., Otmani, A., Tillich, J.-P.: Distinguisher-based attacks on public-key cryptosystems using Reed-Solomon codes. *Des. Codes Cryptogr.* **73**(2), 641–666 (2014)
10. Delsarte, P.: Bilinear forms over a finite field, with applications to coding theory. *J. Comb. Theory Ser. A* **25**(3), 226–241 (1978)
11. Faugère, J.-C., Safey El Din, M., Spaenlehauer, P.-J.: Gröbner bases of Bihomogeneous ideals generated by polynomials of bidegree (1,1): algorithms and complexity. *J. Symb. Comput.* **46**(4), 406–437 (2011)
12. Gabidulin, E.M.: Theory of codes with maximum rank distance. *Probl. Inf. Transm.* (English translation of *Problemy Peredachi Informatsii*) **21**(1), 1–71 (1985)
13. Gabidulin, E.M., Paramonov, A.V., Tretjakov, O.V.: Ideals over a non-commutative ring and their application in cryptology. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 482–489. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_41
14. Gabidulin, E., Rashwan, H., Honary, B.: On improving security of GPT cryptosystems. In: 2009 IEEE International Symposium on Information Theory Proceedings (ISIT), ISIT 2009, pp. 1110–1114 (2009)
15. Gaborit, P., Murat, G., Ruatta, O., Zémor, G.: Low rank parity check codes and their application to cryptography. In: Proceedings of the Workshop on Coding and Cryptography, WCC 2013, Bergen, Norway (2013). www.selmer.uib.no/WCC2013/pdfs/Gaborit.pdf

16. Gaborit, P., Ruatta, O., Schrek, J.: On the complexity of the rank syndrome decoding problem. *IEEE Trans. Inf. Theory* **62**(2), 1006–1019 (2016)
17. Huffman, W.C.: Groups and codes. In: Pless, V.S., Huffman, W.C. (eds.) *Handbook of Coding Theory*. Elsevier, Amsterdam (1998). Chap. 17
18. Loidreau, P.: A new rank metric codes based encryption scheme. In: Lange, T., Takagi, T. (eds.) *PQCrypto 2017*. LNCS, vol. 10346, pp. 3–17. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_1
19. MacWilliams F.J.: Combinatorial properties of elementary Abelian groups Ph.D. thesis, Radcliffe College (1962)
20. McEliece, R.: A public-key cryptosystem based on algebraic coding theory. In: DSN Program Report, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, pp. 114–116, January 1978
21. Misoczki, R., Barreto, P.S.L.M.: Compact McEliece keys from Goppa codes. In: Jacobson, M.J., Rijmen, V., Safavi-Naini, R. (eds.) *SAC 2009*. LNCS, vol. 5867, pp. 376–392. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05445-7_24
22. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.: MDPC-McEliece: new McEliece variants from moderate density parity-check codes. In: *IEEE International Symposium on Information Theory, ISIT 2013*, pp. 2069–2073. IEEE (2013)
23. Otmani, A., Kalachi, H.T., Ndjeya, S.: Improved cryptanalysis of rank metric schemes based on Gabidulin codes. *CoRR* abs/1602.08549 (2016)
24. Overbeck, R.: A new structural attack for GPT and variants. In: Dawson, E., Vaudenay, S. (eds.) *Mycrypt 2005*. LNCS, vol. 3715, pp. 50–63. Springer, Heidelberg (2005). https://doi.org/10.1007/11554868_5
25. Overbeck, R.: Structural attacks for public key cryptosystems based on Gabidulin codes. *J. Cryptol.* **21**(2), 280–301 (2008)
26. Rashwan, H., Honary, B., Gabidulin, E.M.: On improving security of GPT cryptosystems. In: *IEEE International Symposium on Information Theory, ISIT 2009*, pp. 1110–1114. IEEE (2009)
27. Roth, R.M.: Maximum-rank array codes and their application to crisscross error correction. *IEEE Trans. Inf. Theory* **37**(2), 328–336 (1991)

Lightweight Design Choices for LED-like Block Ciphers

Sumanta Sarkar¹(✉), Habeeb Syed¹, Rajat Sadhukhan²,
and Debdeep Mukhopadhyay²

¹ TCS Innovation Labs, Hyderabad, India
{sumanta.sarkar1,habeeb.syed}@tcs.com

² Indian Institute of Technology, Kharagpur, India
rajatssr835@gmail.com, debdeep.mukhopadhyay@gmail.com

Abstract. Serial matrices are a preferred choice for building diffusion layers of lightweight block ciphers as one just needs to implement the last row of such a matrix. In this work we analyze a new class of serial matrices which are the lightest possible 4×4 serial matrix that can be used to build diffusion layers. With this new matrix we show that block ciphers like LED can be implemented with a reduced area in hardware designs, though it has to be cycled for more iterations. Further, we suggest the usage of an alternative S-box to the standard S-box used in LED with similar cryptographic robustness, albeit having lesser area footprint. Finally, we combine these ideas in an end-end FPGA based prototype of LED. We show that with these optimizations, there is a reduction of 16% in area footprint of one round implementation of LED.

Keywords: MDS matrix · Serial matrix · Recursive diffusion layer
Lightweight · S-box · LED

1 Introduction

Lightweight Cryptography is an area that is focused on research and development of cryptographic algorithms suitable for resource constrained devices like RFID tags, wireless sensors, etc. These kind of devices have very low resource, and as such the usual cryptographic algorithms like AES, RSA etc. are not suitable therein. Internet of Things (IoT) is a network of devices like RFIDs/sensors. Therefore, lightweight cryptography plays a crucial role in securing the data that flows in IoT network. IoT has wide applications, for example, health monitoring, supply chain, defense, etc. Thus low area footprint and high throughput are two key areas of focus in lightweight cryptography. Some known lightweight block ciphers include PRESENT [3], PRINCE [4], CLEFIA [20].

Maximum Distance Separable (MDS) matrices are popular choice to build diffusion layers of block ciphers as they have maximal branch number. Block ciphers such as AES, Twofish, SHARK are some of the well known block cipher using MDS matrices for diffusion. For a square matrix to be MDS it needs to

satisfy the condition that its every possible square sub matrix has to be non singular. This requirement makes it challenging to find MDS matrices having efficient implementation in hardware.

In [11] the metric XOR count that measures the implementation cost of a diffusion matrix is introduced. Using this metric one can find MDS matrices which can be efficiently implemented in resource constrained environment. In the recent document produced by NIST [14], the requirement of simpler rounds as a lightweight design principle was emphasized wherein a simple round is iterated over multiple cycles to achieve the desired security. This idea popularized by the lightweight hash function, PHOTON [6] and block cipher, LED [7]. However, it is an open research problem of striving to find further lightweight constructions which can be iterated multiple times to obtain the desired security levels. This method provides an effective mechanism of obtaining lightweight implementations, while not compromising on security, albeit at the cost of extra clock cycles. While for lightweight applications, the gate count of the design is of utmost priority, which can be achieved at the penalty of extra clock cycles, in some applications it may be also a constraint to ensure that the latency does not blow up significantly. This may be important specifically in those environments where energy is also of utmost importance. Hence, it is an interesting research problem of finding lightweight primitives, like linear layers, S-boxes, which can be iterated or cascaded to obtain the same security. On the other hand, the architecture should also amortize the extra latency by employing suitable techniques, which we also strive to find in this work.

In this paper we first explore lightweight recursive MDS layers and show that these diffusion layers can be constructed in terms of serial matrices which have very low XOR count. Known use of 4×4 serial matrices like [6, 7] involve matrices S such that S^4 is MDS. We extend this idea by finding new lightweight serial matrices S for which S^i are MDS for $i > 4$. First we characterize the MDS property of 4×4 serial matrices, where the last row has three 1's (Theorems 1 and 2). We show specific constructions of these lightweight serial matrices (Theorem 2), and show that there exist a matrix with XOR count of 13 (Corollary 1), which is lesser than that of the lightest serial matrix (used for LED), where the XOR count is 16. However, this new matrix needs to be iterated 8 times, while that for LED needs to be repeated for 4 times. In the subsequent part of the paper, we strive to develop a lightweight design of a LED round in hardware, wherein the twice increase in cycles is amortized by a multiple-clock design. In this design, the linear layer which has much lesser critical path compared to the overall critical path (which also includes the S-box), can be operated by a faster clock compared to the overall cipher. Furthermore, we show that ensuring the same cryptographic strength as that of the LED S-box, one can replace with compositions of smaller non-linear S-boxes which have similar robustness, though at a lesser hardware cost. Finally, we combine these ideas and show that the area can be saved by 16% as compared to the original design. However, our design has 30% higher latency. Note that a major application of lightweight encryption is to secure data in IoT network, where low area footprint is always a key factor. requirement of the other lightweight features like throughput, energy, etc., depends

on the applications. For example, if we consider environment monitoring as an IoT application, then we can afford some latency in the encryption algorithm, as this application does not require immediate action upon receiving the data from the environment. However, as the devices are low resourced, it becomes important to decrease the area footprint as low as possible.

Rest of the paper is organized as follows: in Sect. 2 we recall some introductory results on serial matrices, XOR counts followed by Sect. 3 in which we present some new recursive MDS matrices defined by extremely lightweight serial matrices. In Sect. 4 we describe details of implementation of or new primitives in LED block cipher and the resulting optimizations.

2 Preliminaries

Here we briefly recall some basic facts about linear diffusion layers and MDS matrices. Denote by \mathbb{F}_{2^m} finite field of size 2^m . For any $x = (x_0, \dots, x_{n-1}) \in \mathbb{F}_{2^m}^n$ its m -weight $wt_m(x)$ (or simply $wt(x)$ when there no ambiguity) is the count of non zero elements in x . An $n \times n$ linear diffusion layer over m -bit words is a linear map $T : \mathbb{F}_{2^m}^n \rightarrow \mathbb{F}_{2^m}^n$. Diffusion property of T is measured in terms of *differential branch number*, which is defined as

$$BN(T) = \min_{x \in \mathbb{F}_{2^m}^n, x \neq 0} \{wt(x) + wt(Tx)\}.$$

It is well known [5, Chap. 9] that $BN(T) \leq n + 1$ and a diffusion layer that attains the maximum is known as *perfect diffusion layer*¹. Linear diffusion layers are closely connected with MDS codes. Let $\mathcal{C} = [2n, n]$ be a linear code defined over \mathbb{F}_{2^m} . Suppose that $[I|C]$ is a generator matrix of \mathcal{C} , where I is $n \times n$ identity matrix and C is a non singular matrix of the same size. Such a code is MDS if the minimum distance of the code attains Singleton bound [13], i.e., if the minimum distance is $2n - n + 1 = n + 1$. Note that the matrix C can be used to define an $n \times n$ linear diffusion layer over \mathbb{F}_{2^m} and the code \mathcal{C} is MDS if and only if $BN(C) = n + 1$. Extending the notion of MDS codes to matrices, we say that the matrix C is MDS if the code \mathcal{C} is MDS. Another independent way of characterizing an MDS matrix is given below which we will be using to check if a given square matrix is MDS:

Fact 1 [13, Chap. 4]. An $n \times n$ matrix M over \mathbb{F}_{2^m} is MDS if and only if every square submatrix of M is nonsingular.

2.1 XOR Counts

The finite field \mathbb{F}_{2^m} is also a m dimensional vector space over \mathbb{F}_2 . This vector space has several bases but we use only the polynomial basis given by

¹ The term “perfect diffusion layer” was coined by Vaudenay in [22] wherein he suggested for the first time that MDS matrices can be used to design linear diffusion layers.

$\{\alpha, \alpha^2, \dots, \alpha^{m-1}\}$ where α is the root of the irreducible polynomial that defines \mathbb{F}_{2^m} over \mathbb{F}_2 . The notion of XOR count defined below, was introduced in [11] to measure the cost of field multiplication in \mathbb{F}_{2^m} .

Definition 1. Let \mathcal{B} be a vector space basis of \mathbb{F}_{2^m} over \mathbb{F}_2 . For any $a \in \mathbb{F}_{2^m}$ the XOR count of a with respect to \mathcal{B} is denoted by $\text{XOR}(a)$ and is defined as the number of XORs needed to implement the field multiplication of a with an arbitrary element $b \in \mathbb{F}_{2^m}$.

Though the XOR count of a depends on basis of \mathbb{F}_{2^m} , we simply denote it by $\text{XOR}(a)$ whenever there is no ambiguity. Using this metric one can find diffusion matrices which can be efficiently implemented.

The linear diffusion layers in block ciphers are defined by MDS matrices. In [11] the notion of XOR count of an element was extended to XOR count of a row of a matrix. Suppose R_i is a row $R_i = (\beta_{i,0}, \dots, \beta_{i,n-1}) \in \mathbb{F}_{2^m}^n$ of a matrix. Denote by ρ_i the number of non zero entries in R_i , then the XORs needed to implement row R_i is given by

$$\sum_{j=0}^{n-1} \text{XOR}(\beta_{i,j}) + (\rho_i - 1) \cdot m. \tag{1}$$

This notion of XOR count was further extended to the full matrix in [18] as

$$\text{XOR}(M) = \sum_{i=0}^n \sum_{j=0}^{n-1} \text{XOR}(\beta_{i,j}) + \sum_{i=0}^n (\rho_i - 1) \cdot m, \tag{2}$$

where M is an $n \times n$ matrix over \mathbb{F}_{2^m} . In this paper we are mainly focused on Serial matrices, in which we only need to know the cost of the last row (1).

Based on the notion of XOR count several works followed [12, 17, 21] in order to obtain MDS matrices with low XOR counts. For instance, [18] showed the minimum value of XOR count that 4×4 MDS matrices can have over \mathbb{F}_{2^4} and \mathbb{F}_{2^8} . Recently [19] presented 8×8 MDS matrices with the lowest known XOR counts over \mathbb{F}_{2^4} and \mathbb{F}_{2^8} .

3 Recursive MDS Matrices

A serial matrix of order $n \times n$ over \mathbb{F}_{2^m} is a matrix of the form

$$S = \begin{bmatrix} 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 1 \\ a_0 & a_1 & \dots & a_{n-1} \end{bmatrix}, \tag{3}$$

which is usually denoted by $S = \text{Serial}(a_0, \dots, a_{n-1})$. The matrix S is companion matrix of the monic polynomial $a_0 + a_1X + \dots + a_{n-1}X^{n-1} + X^n \in \mathbb{F}_{2^m}[X]$ with $a_0 \neq 0$. One can easily see that inverse of S is

$$S^{-1} = \begin{bmatrix} \frac{a_1}{a_0} & \frac{a_2}{a_0} & \dots & \frac{a_{n-1}}{a_0} & \frac{1}{a_0} \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}. \tag{4}$$

By a recursive MDS matrix we mean a MDS matrix M such that $M = S^i$ for some serial matrix S and positive integer i . Recursive MDS matrices are a preferred choice for diffusion layers in lightweight cryptography as only the last row of such a matrix needs to be implemented. However, the downside is that it causes latency since the output of diffusion layer is obtained by applying S recursively as

$$(y_0, \dots, y_{n-1}) = \underbrace{S \dots S}_{i \text{ times}}(x_0, \dots, x_{n-1}) \dots. \tag{5}$$

An additional advantage of using recursive MDS matrix in block cipher is that its inverse also has simple form (as in (4)) and can be implemented efficiently.

Several techniques have been proposed to construct recursive diffusion layers. In [1, 16, 23] authors presented construction of recursive diffusion layers using binary linear maps instead of matrices defined over \mathbb{F}_{2^m} . Later Augot and Finiasz constructed recursive MDS matrices from shortened BCH codes [2] following which more general characterization of Recursive MDS matrices is presented in [8, 9]. In [10] authors discussed construction of 4×4 MDS matrices from serial matrices defined over sets of the form $\{1, \alpha, \alpha^2, \alpha + 1\} \subset \mathbb{F}_{2^m}$.

3.1 Lightweight Recursive MDS Matrices

If $S = \text{Serial}(a_0, \dots, a_{n-1})$ is an $n \times n$ serial matrix defined over \mathbb{F}_{2^m} then it is easy to see that the least possible value of i for which S^i is MDS is $i = n$. Consequently while constructing such MDS matrices the usual practice is to find an $n \times n$ serial matrices S for which S^n is MDS. This is done mainly to minimize throughput latency: If S^i is MDS then we need i iterations to compute the output $y = (S^i) \cdot x$ (see (5)) and hence optimal throughput is achieved when $i = n$. However, it is possible to find new lightweight serial matrices S such that S^i is MDS if we assume $i \geq n$.

In this section we present some new lightweight recursive MDS matrices which have not been analyzed so far. To begin we briefly recall some terminology from [2] which will be useful in presenting new results. Suppose $S = \text{Serial}(a_0, \dots, a_{n-1})$ be the companion matrix of the polynomial $f(X) = a_0 + a_1X + \dots + a_{n-1}X^{n-1} + X^n$ defined over \mathbb{F}_{2^m} . We can interpret the matrix S as

$$S = \begin{bmatrix} 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 1 \\ a_0 & a_1 & \dots & a_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} X \\ X^2 \\ \vdots \\ X^{n-1} \\ X^n \bmod f(X) \end{bmatrix}}_{(*)} \tag{6}$$

where elements of each row in the matrix (*) consists of coefficients of the polynomial given in that row. With these notations it is easy to see that for any $i \geq 1$ the matrix S^i is given by

$$S^i = \begin{bmatrix} X^i \bmod f(X) \\ X^{i+1} \bmod f(X) \\ \vdots \\ X^{i+(n-1)} \bmod f(X) \end{bmatrix}. \tag{7}$$

Recall that in general one is interested in $n \times n$ serial matrices S for which S^n is MDS in order to optimize the throughput. However in many cases it is not possible to obtain such MDS matrices. In the following we identify two such classes.

Lemma 1. *Let $n \geq 4$ and $S = \text{Serial}(a_0, \dots, a_{n-1})$, be defined over \mathbb{F}_{2^m} . Then S^n is not MDS if $a_{n-1} = a_{n-2} = 1$ or $a_{n-1} = a_{n-3} = 1$.*

Proof. Suppose $f(X) = a_0 + \dots + a_{n-1}X^{n-1} + X^n$ be the the polynomial associated with the matrix S . We define c, c_i as follows. Let $c = a_{n-1}^2 + a_{n-2}$ and for $i = 0, \dots, n - 3$,

$$c_i = a_i \cdot a_{n-1} + a_{i-1}, \tag{8}$$

where we use the convention that $a_i = 0$ for $i < 0$. Using these notations one can check that

$$X^{n+2} \bmod f(X) = a_0 \cdot c + \sum_{i=1}^{n-2} (a_i c + c_{i-1})X^i + (a_{n-1}^3 + a_{n-3})X^{n-1}. \tag{9}$$

From (7) it follows that the coefficients occurring in above polynomial form third row in the matrix S^n . If $a_{n-1} = a_{n-2} = 1$ then we get that $c = 0$ and consequently the matrix S^n is not MDS. Similarly if $a_{n-1} = a_{n-3} = 1$ then the coefficient of X^{n-1} in (9) becomes zero and the matrix S^n is not MDS. \square

Lemma 2. *Let $S = \text{Serial}(a_0, \dots, a_{n-1})$ be a matrix defined over \mathbb{F}_{2^m} . Then S^n is not MDS if $a_{i-1} = a_i = a_{n-1} = 1$ for any $i = 1, \dots, n - 2$*

Proof. Using c_i as in (8) we have

$$X^{n+1} \bmod f(X) = a_0 a_{n-1} + \sum_{i=1}^{n-2} c_i X^i + (a_{n-1}^2 + a_{n-2})X^{n-1} \tag{10}$$

coefficients of which occur as second row in the matrix S^n . Here we have defined c_{n-2} precisely as in (8). If $a_{i-1} = a_i = a_{n-1} = 1$ for any $i = 1, \dots, n - 2$ then $c_i = 0$ and hence the matrix S^n is not MDS. \square

While searching for lightweight recursive MDS matrices one would like to consider the lightest possible serial matrix, $S = \text{Serial}(1, \dots, 1)$ which consists of only ‘1’ as entries. However, one can easily check that in this case S^i is not MDS for any $i \geq 1$. We state this observation as a fact:

Fact 2. Let $S = \text{Serial}(1, \dots, 1)$ be an $n \times n$ be defined over \mathbb{F}_{2^m} . Then S^i is not MDS for any $i \geq 1$.

Following Fact 2, the lightest possible recursive matrix could be of the form $S = \text{Serial}(a_0, \dots, a_{n-1})$ where $a_i \neq 1$ for some $0 \leq i \leq n - 1$ and $a_j = 1$ for every $0 \leq j \neq i \leq n - 1$. Our objective is to find the lightest possible such matrix for which S^i is MDS with the minimum $i \geq n$.

In practice size of the diffusion matrix used in a lightweight block cipher is 4×4 . Keeping this in mind we fix $n = 4$ in the remaining part of this Section. In [10] authors show that if $S = \text{Serial}(a_0, a_1, a_2, a_3)$ is such that $a_i = 1$ for more than 2 values of i then S^4 is never MDS over \mathbb{F}_{2^m} . If we relax the condition that S^4 need to be MDS and consider matrices such that S^i is MDS for $i > 4$, then we get new serial matrices which are the lightest possible. In the following we analyze MDS property of S^i , where $S = \text{Serial}(a_0, a_1, a_2, a_3)$ such that $a_i = 1$ for 3 values of i .

Theorem 1. Let $S = \text{Serial}(a_0, a_1, a_2, a_3)$ be a serial matrix defined over \mathbb{F}_{2^m} in which $a_i = 1$ for precisely 3 values of i . For $1 \leq i \leq 8$ the matrices S^i are not MDS if

- (i) $S = \text{Serial}(a, 1, 1, 1)$
- (ii) $S = \text{Serial}(1, a, 1, 1)$
- (iii) $S = \text{Serial}(1, 1, 1, a)$,

where $a \notin \{0, 1\}$.

Proof. Let $S = \text{Serial}(a_0, a_1, a_2, a_3)$ be a serial matrix defined over \mathbb{F}_{2^m} . From (7) it follows directly that for $i = 1, 2, 3$ the matrix S^i has 0 entries and hence cannot be MDS. Remains to show that S^i is not MDS for $4 \leq i \leq 8$ whenever S is in any of the form (i), (ii), (iii) as given in theorem. We do this by considering each form separately. In the following we denote the polynomial associated with the serial matrix S by $f(X)$.

Case 1. $S = \text{Serial}(a, 1, 1, 1)$, $a \notin \{0, 1\}$.

The polynomial corresponding to the serial matrix S is $f(X) = a + X + X^2 + X^3 + X^4$ from which it follows that

$$X^7 \bmod f(X) = 0 + 0 \cdot X + a X^2 + (a + 1) X^3.$$

Note that this polynomial has zero coefficients and that these coefficients form a row in the matrices S^i for $4 \leq i \leq 7$ as can be seen from (7). Consequently none

of these matrices can be MDS. Using same argument we see that the matrix S^8 cannot be MDS because the coefficients of

$$X^{10} \bmod f(X) = a^2 + 0 \cdot T + (a^2 + 1) \cdot T^2 + 0 \cdot T^3.$$

form a row in this matrix.

Case 2. $S = \text{Serial}(1, a, 1, 1)$, $a \notin \{0, 1\}$.

In this case we see that

$$X^7 \bmod f(X) = (a + 1) + (a^2 + a) \cdot X + a \cdot X^2 + 0 \cdot X^3, \text{ and} \tag{11a}$$

$$X^8 \bmod f(X) = 0 + (a + 1) \cdot X + (a^2 + a) \cdot X^2 + a \cdot X^3. \tag{11b}$$

Using the argument as in Case 1 we see that the matrices S^i cannot be MDS for $4 \leq i \leq 7$ because of zero coefficients in (11a) and the matrix S^8 cannot be MDS because of zero coefficients in (11b).

Case 3. $S = \text{Serial}(1, 1, 1, a)$, $a \notin \{0, 1\}$.

Unlike previous cases, here all the matrices S^i contain non zero entries for $4 \leq i \leq 8$. However each of this matrix has a 2×2 singular submatrix. To prove this first note that

$$X^7 \bmod f(X) = a^3 + 1 + (a^3 + a^2)X + (a^3 + a^2 + a)X^2 + (a^4 + a^2)X^3 \tag{12a}$$

$$X^8 \bmod f(X) = a^4 + a^2 + (a^4 + a^3 + a^2 + 1)X + (a^4 + a^3)X^2 + (a^5 + a^2 + a)X^3 \tag{12b}$$

For $i \geq 1$ let $R_i = (r_{i,0}, r_{i,1}, r_{i,2}, r_{i,3})$ where $r_{i,j}$ is the Coefficient of X^j in the polynomial $(X^i \bmod f(X))$. Using (7) we know that R_7, R_8 occur as two consecutive rows in the matrices S^i for $i = 5, 6, 7$. From (12a) and (12b) it is easy to see that $r_{7,0}r_{8,2} + r_{8,0}r_{7,2} = 0$ which implies that the matrices S^i have a 2×2 singular submatrix for $i = 5, 6, 7$ and hence are not MDS matrices. Finally it remains to show that S^8 is also not MDS. We have

$$X^{10} \bmod f(X) = (a^6 + a^4 + a^2) + (a^6 + a^5 + a^4 + a)X + (a^6 + a^5 + a^2 + a)X^2 + (a^7 + a^4 + a + 1)X^3 \tag{13}$$

and

$$X^{11} \bmod f(X) = a^7 + a^4 + a + 1 + (a^7 + a^6 + a^2 + a + 1)X + (a^7 + a^6 + a^5 + 1)X^2 + (a^8 + a^6)X^3 \tag{14}$$

The rows R_{10}, R_{11} occur in the matrix S^8 and from (13) and (14) we see that the sub matrix

$$\begin{bmatrix} r_{10,1} & r_{10,2} \\ r_{11,1} & r_{11,2} \end{bmatrix} \tag{15}$$

is a singular submatrix of S^8 making it non MDS. □

Theorem 2. *Let $S = \text{Serial}(1, 1, a, 1)$ be defined over \mathbb{F}_{2^m} , then S^i is not MDS for $1 \leq i \leq 7$. Further, S^8 is MDS precisely in either the following two cases:*

- (1) If the minimal polynomial of a over \mathbb{F}_2 is $X^4 + X + 1$
- (2) If the degree of the minimal polynomial of a over \mathbb{F}_2 is ≥ 5 and the minimal polynomial is not in the set $\{X^5 + X^4 + X^2 + X + 1, X^5 + X^3 + X^2 + X + 1, X^5 + X^4 + X^3 + X^2 + 1\}$

Proof. Let $f(x) = 1 + x + ax^2 + x^3 + x^4$ be the associated polynomial of the serial matrix $S = \text{Serial}(1, 1, a, 1)$ from which we can easily see that

$$x^7 \bmod f(x) = 0 + (a + 1)x + ax^2 + (a^2 + a)x^3.$$

Using the argument as in Case 1 of Theorem 1 we conclude that S^i is not MDS for $1 \leq i \leq 7$.

To prove the remaining part of the theorem denote by $S(x)$ the matrix of the form $\text{Serial}(1, 1, x, 1)$ where x is an indeterminate. Let Δ_i be the set of determinants of all of the $i \times i$ submatrices of $S^8(x)$. We have

$$\begin{aligned} \Delta_1 &= \{x^3 + x^2 + x, x^3 + x^2 + x + 1, x^4 + 1, x^2, x^2 + 1, x, \\ &\quad x^4 + x^2, x^2 + x, x^3 + x + 1\} \\ \Delta_2 &= \{x^6 + x^5 + x + 1, x^5 + x^4 + x, x^5 + x^3 + x^2 + 1, x^5 + x^3 + x, \\ &\quad x^6 + x^4 + 1, x^6 + x^2, x^4 + x^2, x^4 + x^3, x^7 + x^6 + x^5 + x^3 + x + 1, \\ &\quad x^5 + x, x^6 + x^4 + x^3 + x^2 + x, x^6, x^4 + 1, x^6 + x^5 + x^4 + x, \\ &\quad x^7 + x^5 + x^3 + x, x^5 + x^4 + x^3 + x, x^8 + x^6 + x^2 + 1, x^7 + x^6 + x^3 + 1\} \\ \Delta_3 &= \{x^3 + x^2 + x, x^3 + x^2 + x + 1, x^4 + 1, x^2, x^2 + 1, x, x^4 + x^2, x^2 + x, x^3 + x + 1\} \\ &\quad \text{and } \Delta_4 = \{1\}. \end{aligned}$$

Denote by Δ the set of irreducible factors of polynomials in $\Delta_1 \cup \Delta_2 \cup \Delta_3$. It is easy to see that,

$$\begin{aligned} \Delta &= \{x, x + 1, x^2 + x + 1, x^3 + x + 1, x^3 + x^2 + 1, \\ &\quad x^4 + x^3 + 1, x^4 + x^3 + x^2 + x + 1, \\ &\quad x^5 + x^4 + x^2 + x + 1, x^5 + x^3 + x^2 + x + 1, x^5 + x^4 + x^3 + x^2 + 1\} \end{aligned}$$

Now, if we consider the matrix $S(a)$ for some $a \in \mathbb{F}_{2^m}$ then $S(a)^8$ is MDS if and only if $\delta(a) \neq 0$ for every $\delta(x) \in \Delta$. One can check that this happens precisely in either the two cases (1), (2) stated in statement of theorem. \square

Corollary 1. *The matrix $S = \text{Serial}(1, 1, \alpha, 1)$ defined over \mathbb{F}_{2^4} , where α is a root of irreducible polynomial $X^4 + X + 1$ is the lightest serial matrix such that S^8 is MDS, and $\text{XOR}(S) = 13$.*

Proof. As $\text{Serial}(1, 1, 1, 1)$ can never be MDS for any $(\text{Serial}(1, 1, 1, 1))^i$, thus the next possibility that $(\text{Serial}(a_0, a_1, a_2, a_3))^8$ is MDS when it is of the form $\text{Serial}(1, 1, a, 1)$ for some $a \notin \{0, 1\}$. The element α which is a root of $X^4 + X + 1 = 0$ has the lowest nonzero XOR count which is 1. This results in that S is the lightest serial matrix such that S^8 is MDS, and $\text{XOR}(S) = 13$. \square

4 Lightweight Architecture for Block Ciphers: A Case Study on LED

In this section, we will describe the hardware implementation of datapath of typical AES-like block ciphers. We consider the particular instance of LED, though the idea can be generalized for a larger class of block ciphers having an MDS block as diffusion layer. Along with implementing the above discussed lightweight linear layer, we focus on two other important aspects: (1) the design of the non-linear S-box, and (2) the overall composition of the layers to ensure that the two-fold increase in the iteration requirement of the modified linear layer does not lead to a double increase in the latency. We start with discussion on the the non-linear S-box.

We have implemented the datapath of LED, using an ASIC design flow. We have used *Synopsys Design Compiler (version: vI-2013.12-SP5-4)* for synthesis and *Synopsys VCS (version: I-2014.03-SP1-1)* for simulation. Standard cell library (*TSL18FS120*) on 180 nm technology from *TowerSemiconductor Ltd.* is used during synthesis, which is characterized using *SiliconSmart Software (version: 2008.02-SP1p1)* under *Fast-Fast process (P)*, *1.98V voltage (V)* and *-40°C temperature (T)*.

4.1 Choosing an Efficient 4 × 4 S-box

The 4 × 4 S-box that is used in LED has nonlinearity 4, differential uniformity 4, and algebraic degree 3. The polynomial expression over \mathbb{F}_{2^4} of this S-box is

$$\begin{aligned}
 &(\alpha^3 + \alpha^2 + 1)x^{14} + (\alpha^3 + \alpha^2 + 1)x^{13} + (\alpha^3 + \alpha^2)x^{12} + (\alpha^3 + \alpha^2 + \alpha)x^{11} \\
 &\quad + (\alpha^3 + 1)x^{10} + (\alpha^3 + 1)x^9 + (\alpha^2 + \alpha + 1)x^8 + \alpha^2x^7 + (\alpha^3 + \alpha^2)x^6 \\
 &+ (\alpha^3 + \alpha)x^5 + (\alpha^3 + \alpha^2 + \alpha)x^4 + (\alpha^2 + \alpha + 1)x^3 + (\alpha^2 + \alpha + 1)x^2 + \alpha^3 + \alpha^2,
 \end{aligned}$$

where α is a primitive root of $X^4 + X + 1 = 0$. Instead an S-box which is monomial would have low hardware footprint. The monomials $X \mapsto X^i$ for $i = 7, 11, 13, 14$, are such that the associated 4 × 4 S-box has nonlinearity 4, differential uniformity 4, and algebraic degree 3. We consider such monomial with the least i , i.e., $X \mapsto X^7$. As this S-box has fixed points: $0 \mapsto 0, 1 \mapsto 1$, etc., we consider $X \mapsto X^7 + 1$. The associated S-box will thus not have any fixed points, while the other cryptographic properties like nonlinearity, differential uniformity, degree remain invariant. The proposed S-box values are shown in Table 1.

We also evaluate that the implementation cost of this S-box is lesser than that of LED. We implement the proposed S-box function as $X^7 = X^4 \times X^2 \times X$, which

Table 1. S-box defined by $X \mapsto X^7 + 1$

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	1	0	A	C	8	F	7	6	D	4	9	2	E	3	5	B

Table 2. S-box properties as evaluated by S-box evaluation tool

Property	Proposed S-box	LED S-box
Nonlinearity	4	4
Algebraic degree	3	3
Algebraic immunity	2	2
Differential uniformity	4	4
Robustness to differential cryptanalysis	0.750	0.750
Silicon area	359 μm^2	391 μm^2

requires 2 nibble wise multiplication operation, 1 square and 1 fourth power calculation followed by one bit XOR. We did not choose $X^7 = X^4 \times X^3$, as X^3 is heavier than the 2 multiplication operations in case of the former decomposition. Our proposed 4×4 S-box occupies $359 \mu\text{m}^2$ area, which is 28.6 GE. Here 1 GE is the area required for one 2-input NAND gate. A 2-input lowest drive NAND gate of our library occupies $12.544 \mu\text{m}^2$ area. This design may be compared with the standard look-up table based LED S-box implementation occupies $391 \mu\text{m}^2$, which is 31.2 GE, which is 9% more than the proposed S-box. We compare the two designs with respect to both cryptographic strengths and area requirement using the S-Box Evaluation Tool (SET) [15] tool in Table 2. It may be observed that we have reduced the hardware overhead of the non-linear layer using our proposed method without compromising on security parameters like non-linearity, differential uniformity and degree of a typical 4×4 S-box as used in LED.

4.2 Implementing Lightweight Serial Matrix

In this section we describe the implementation strategy of our new lightweight MDS matrix and compare it with existing MDS matrix used in LED. Denote by S_1 the serial matrix $\text{Serial}(\alpha^2, 1, \alpha, \alpha)$ which is the existing diffusion matrix of LED block cipher. This matrix is considered to be lightest which has $\text{XOR}(S_1) = 16$ with S_1^4 being MDS. We implemented $S_1 \times X$, where $X = [x_1, x_2, x_3, x_4]^t$ is a column vector, each element x_i of the vector is a nibble. This implementation has hardware footprint $387.5 \mu\text{m}^2$ (31 GE) comprising of six 3-input XOR gates and two 2-input XOR gates.

Next consider the serial matrix $S_2 = \text{Serial}(1, 1, \alpha, 1)$ as given in Corollary 1. This serial matrix is lighter than S_1 with XOR cost 13, using which MDS matrix is calculated as S_2^8 . Similar to the implementation of S_1 , we implemented $S_2 \times X$, and hardware footprint reported by the synthesis tool is $365.4 \mu\text{m}^2$ (29 GE) comprising of five 3-input XOR gates and three 2-input XOR gates.

We design the entire data-path of LED using proposed transformation and the overall design results show compaction. The critical path length and the overall area of the linear layer and the entire data-path is shown in Table 3 below. From the table it can be seen that there is saving of 16% area in the proposed datapath

Table 3. Area and critical path time comparison for one round implementation of our design and LED

Design	$Cr_{datapath}$ (ns) ^a	$Cr_{linearlayer}$ (ns)	$A_{datapath}$ (GE)	$A_{linearlayer}$ (GE)
LED	3.08	0.61	1244.2	123.56
Our design	2.24	0.61	1044.6	116.5

^aThis latency is due to one iteration of serial matrix along with S-box implementation and shift row operation.

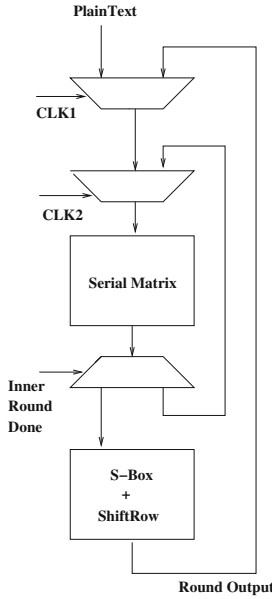


Fig. 1. Datapath design of our proposed method

(1044.6 GE) compared to the original LED datapath (1244.2 GE). However, one may argue that in the proposed linear layer result obtained by computing $S_2^8 \times X$, (X is the state matrix) requiring 8 clock cycles, whereas in the original LED design the linear layer result is obtained from $S_1^4 \times X$ which requires only 4 clock cycles. So it may seem that the new design incurs twice latency compared to the original design for one round implementation. Our proposed S-box takes lesser time to execute, notably, delay for the S-box and shift row combined is 2.47 ns for LED, whereas for the new design it is just 1.63 ns. This helps reduce the overall delay in our design not reaching the double the delay of LED. Then overall latency of the LED is $0.61 \times 4 + 2.47 = 4.91$ ns, while that for the new design it is $0.61 \times 8 + 1.63 = 6.51$ ns. Thanks to lower latency incurred by the S-box, the overall increase in latency of the new design is capped at 30%.

In IoT applications low area footprints is always a factor, whereas in some applications like environment monitoring some latency is affordable. So, for these class of applications our design has very high impact.

4.3 Restraining Latency by Double Clock Architecture

The new lightweight MDS matrix S_2 requires 8 iterations to compute the full effect of diffusion layer since S_2^8 is MDS. Compared to original matrix S_1 which needs 4 iterations the matrix S_2 increases latency. We now describe a solution for reducing the latency caused by our serial matrix implementation. If the user wants to have very low area footprint like our design, and also wants to have low latency, then following is our suggestion. One can use double clock architecture to check the latency of the new design. We present this in Fig. 1. The figure shows the operation of the serial matrix is done at a clock clk_2 which is faster than clk_1 , rests of the operations are done at clk_1 . With this architecture we can curb the latency and at the same time can benefit from the low area cost.

5 Conclusions

Latency is inherent to the block ciphers whose diffusion layer is based on serial matrices. In this work we have shown that if we relax latency slightly, then we can further reduce the implementation cost of the diffusion layer. We have applied our newly discovered matrix in the diffusion layer of LED, and on top of that we also have proposed a lighter S-box. The combining effect of these two is that we have obtained a variant of LED which is lighter than the original one. We also have proposed a multi-clock design of the LED data-path which can be used to restrain increase in the latency.

Our diffusion matrix opens up the applicability of an $n \times n$ serial matrix S in lightweight block ciphers, such that S^i is MDS for $i > n$. On the other hand, the proposed multi-clock architecture is also interesting to explore further.

References

1. Augot, D., Finiasz, M.: Exhaustive search for small dimension recursive MDS diffusion layers for block ciphers and hash functions. In: 2013 IEEE International Symposium on Information Theory Proceedings (ISIT), pp. 1551–1555. IEEE (2013)
2. Augot, D., Finiasz, M.: Direct construction of recursive MDS diffusion layers using shortened BCH codes. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 3–17. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46706-0_1
3. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: 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

4. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knežević, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE - a low-latency block cipher for pervasive computing applications (Full version). Cryptology ePrint Archive, Report 2012/529 (2012). <http://eprint.iacr.org/>
5. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer, Heidelberg (2002)
6. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Berlin, Heidelberg (2011)
7. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23951-9_22
8. Gupta, K.C., Pandey, S.K., Venkateswarlu, A.: On the direct construction of recursive MDS matrices. Des. Codes Crypt. **82**(1–2), 77–94 (2017)
9. Gupta, K.C., Pandey, S.K., Venkateswarlu, A.: Towards a general construction of recursive MDS diffusion layers. Des. Codes Crypt. **82**(1–2), 179–195 (2017)
10. Gupta, K.C., Ray, I.G.: On constructions of MDS matrices from companion matrices for lightweight cryptography. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CD-ARES 2013. LNCS, vol. 8128, pp. 29–43. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40588-4_3
11. Khoo, K., Peyrin, T., Poschmann, A.Y., Yap, H.: FOAM: Searching for hardware-optimal SPN structures and components with a fair comparison. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 433–450. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_24
12. Liu, M., Sim, S.M.: Lightweight MDS generalized circulant matrices. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 101–120. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_6
13. Macwilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes (North-Holland Mathematical Library). North Holland, January 1983
14. McKay, K.A., Bassham, L., Turan, M.S., Mouha, N.: NISTIR 8114, Report on Lightweight Cryptography (2017). <http://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf>
15. Picek, S., Batina, L., Jakobović, D., Ege, B., Golub, M.: S-box, SET, match: A toolbox for S-box analysis. In: Naccache, D., Sauveron, D. (eds.) WISTP 2014. LNCS, vol. 8501, pp. 140–149. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43826-8_10
16. Sajadieh, M., Dakhilalian, M., Mala, H., Sepehrdad, P.: Recursive diffusion layers for block ciphers and hash functions. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 385–401. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_22
17. Sarkar, S., Sim, S.M.: A deeper understanding of the XOR count distribution in the context of lightweight cryptography. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2016. LNCS, vol. 9646, pp. 167–182. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31517-1_9
18. Sarkar, S., Syed, H.: Lightweight diffusion layer: Importance of toeplitz matrices. IACR Trans. Symmetric Cryptol. **2016**(1), 95–113 (2016)
19. Sarkar, S., Syed, H.: Analysis of toeplitz MDS matrices. Cryptology ePrint Archive, Report 2017/368 (2017). <http://eprint.iacr.org/2017/368>

20. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit block-cipher CLEFIA (Extended Abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74619-5_12
21. Sim, S.M., Khoo, K., Oggier, F., Peyrin, T.: Lightweight MDS involution matrices. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 471–493. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_23
22. Vaudenay, S.: On the need for multipermutations: Cryptanalysis of MD4 and SAFER. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 286–297. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60590-8_22
23. Wu, S., Wang, M., Wu, W.: Recursive diffusion layers for (Lightweight) block ciphers and hash functions. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 355–371. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_23

Looting the LUTs: FPGA Optimization of AES and AES-like Ciphers for Authenticated Encryption

Mustafa Khairallah¹^(✉), Anupam Chattopadhyay^{1,2},
and Thomas Peyrin^{1,2}

¹ School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore, Singapore
mustafam001@e.ntu.edu.sg

² School of Computer Science and Engineering,
Nanyang Technological University, Singapore, Singapore
{anupam,thomas.peyrin}@ntu.edu.sg

Abstract. In this paper, we investigate the efficiency of FPGA implementations of AES and AES-like ciphers, specially in the context of authenticated encryption. We consider the encryption/decryption and the authentication/verification structures of OCB-like modes (like OTR or SCT modes). Their main advantage is that they are fully parallelisable. While this feature has already been used to increase the throughput/performance of hardware implementations, it is usually overlooked while comparing different ciphers. We show how to use it with zero area overhead, leading to a very significant efficiency gain. Additionally, we show that using FPGA technology mapping instead of logic optimization, the area of both the linear and non linear parts of the round function of several AES-like primitives can be reduced, without affecting the run-time performance. We provide the implementation results of two multi-stream implementations of both the LED and AES block ciphers. The AES implementation in this paper achieves an efficiency of 38 Mbps/slice, which is the most efficient implementation in literature, to the best of our knowledge. For LED, achieves 2.5 Mbps/slice on Spartan 3 FPGA, which is 2.57x better than the previous implementation. Besides, we use our new techniques to optimize the FPGA implementation of the CAESAR candidate Deoxys-I in both the encryption only and encryption/decryption settings. Finally, we show that the efficiency gains of the proposed techniques extend to other technologies, such as ASIC, as well.

Keywords: AES · FPGA · Authenticated encryption
Logic optimization · Technology mapping · Deoxys · LED

1 Introduction

In September 2016, the CAESAR competition committee announced the selection of 15 Authenticated Encryption with Associated Data (AEAD) schemes as

candidates for round 3 of the CAESAR competition [1]. This competition signifies the current need for practical, secure and efficient AEAD schemes. An AEAD scheme typically consists of two routines. The first one is encryption $\mathcal{E}_K(AD, M)$ which takes as input a shared key K , public associated data AD and the message to be encrypted M and returns a tagged ciphertext C . The second one is decryption/verification $\mathcal{D}_K(AD, C)$, which either returns an invalid symbol \perp if the received ciphertext, associated data and the authentication data do not match, or the decrypted message M , otherwise.

An important aspect of the study of AEAD schemes is the evaluation of their hardware performance, which clearly needs more efforts. So far, nearly all candidates have been supported with a basic hardware implementation [2]. However, the implementations are done on various platforms, for different interfaces and thus, a comprehensive evaluation is still missing. Furthermore, several designs have unique advantages to offer in some platforms, e.g., Field Programmable Gate Array (FPGA), which is not fully exploited. This is one of the prime motivations for this manuscript.

In the survey presented in [3], the authors classified the round 2 candidates of the CAESAR competition into five families according to their base constructions: block cipher-based, stream cipher-based, key-less permutations, hash-function-based and dedicated schemes. In this paper, we focus on the block cipher-based family. Specifically, we focus on optimizations for algorithms that allow block-level parallelism while using the underlying block cipher, such as the Offset Code Book mode (OCB) [4–6], the Synthetic Counter-in-Tweak mode (SCT) [7] and the Offset Two-Round mode (OTR) [8].

All the available hardware implementations of the CAESAR competition candidates on the ATHENA hardware evaluation website [2] are fully sequential implementations, i.e. to start processing a new block, all the previous blocks have to be finished. These implementations do not take full advantage of the specific characteristics of the schemes based on the aforementioned modes.

Generally, circuit optimization consists of two phases: logic synthesis and technology mapping. For certain target technologies, such as FPGA, logically optimized circuits do not provide the optimal mapping to the underlying technology, leaving behind a lot of under-utilized hardware resources. This phenomenon is obvious in the AES Sbox circuits proposed by Boyar [9, 10], which are logical optimizations of the circuit proposed by Canright [11]. These circuits are much smaller than the straight-forward ROM-based Sbox in terms of gate count and circuit depth. These two features make them the natural choice for low area Application-Specific Integrated Circuits (ASIC) implementations of AES. Interestingly, on the other hand, practical results show that one can achieve a smaller area on FPGA by using the ROM approach [12]. By analyzing this result, it appears that due to the specific details of these circuits [9–11], it is hard to map them efficiently to look-up tables (LUTs) that the FPGA is constructed from, leading to a lot of under-utilized/unusable logic gates inside the FPGA.

In Fig. 1, the number of LUTs required for implementing two 8-bit to 8-bit ROM-based Sboxes (which are both the forward and inverse AES Sboxes)

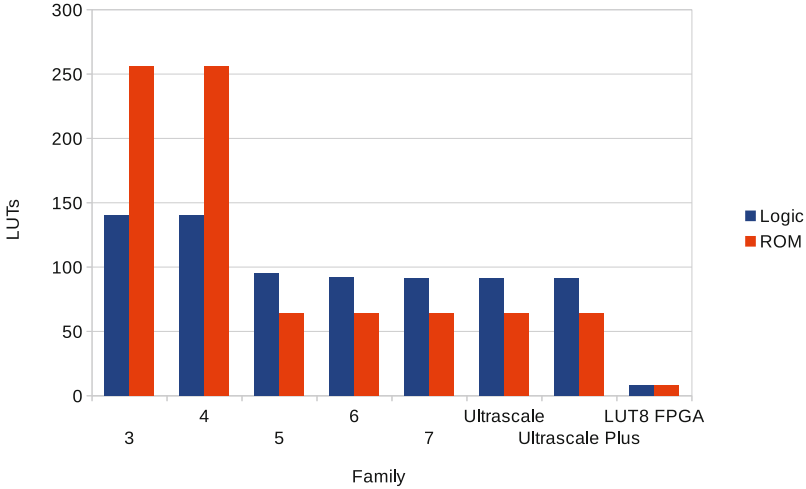


Fig. 1. Evolution of the AES Sbox/ISbox area vs. Xilinx FPGA families

is compared with the implementation of Boyar’s shared encryption/decryption Sbox [9]. These results are plotted against the technology evolution of Xilinx FPGAs as an example. Analysing the chart, it is clear that after the introduction of the Virtex 5 family, logic optimization of the Sbox stopped being beneficial. The reason for that was the introduction of the 6-input LUTs, which enabled implementing an 8-to-1 look-up table using only four 6-input LUTs and three dedicated multiplexers, or five 6-input LUTs. In other words, the ROM-based Sbox has become both faster and smaller than the logic-based Sbox, even when both encryption and decryption are implemented using a shared data path. While the technology seems to be saturated around the 6-input LUT structure, a hypothetical family has been added to the chart assuming 8-input LUT structure, showing that such a family will make the cost of both logic-based and ROM-based implementations exactly the same (8 LUTs). While these results may seem specific to Xilinx FPGAs, other vendors, e.g. Altera, also use 6-input LUTs as their building blocks and will follow the same trend. Besides, the FPGA industry seems to be saturated around this building block and we believe that the same trend will follow for the upcoming years.

In a nutshell, the current implementations for multiple designs in the CAE-SAR contest do neither exploit the underlying block-level parallelism and nor consider the FPGA-specific optimizations. Both of these shortcomings are addressed in the current manuscript, achieving significant gain in area-efficiency, run-time performance or both.

Our Contributions. In this article, we propose new improvements for FPGA implementations of AEAD schemes based on AES-like primitive. These improvements are twofold.

Firstly, we provide a new efficient hardware architecture for OCB-like AEAD modes (Sect. 2). The architecture uses a generic multi-stream AES-like cipher, such as AES or Deoxys-BC (the tweakable block cipher used in CAESAR competition candidate Deoxys [13]) as an underlying primitive. This architecture can be easily modified to support the OTR or SCT AEAD modes for example.

Secondly, we improve the implementation efficiency of several AES-like ciphers, such as AES, LED and Deoxys-BC. In particular, the problem of FPGA mapping and under-utilized hardware discussed earlier is studied in details for two applications (Sect. 3):

- we show how to design low-area logic primitives optimized for FPGA LUTs instead of the number of logic gates (Sect. 3.2).
- we explain how to select the locations of pipelining registers to accommodate as many independent streams as possible without any additional area cost compared to the single stream architecture (Sect. 3.3).

Eventually, as practical results, following these implementation strategies we obtained very efficient LED and AES implementations (Sect. 4). For example, our AES implementation achieves an efficiency of 38 Mbps/slice, which is the most efficient AES FPGA implementation in the literature to the best of our knowledge. We also applied our techniques to Deoxys, and we obtained the current best Deoxys-I FPGA implementation, improving their efficiency by a factor ~ 1.7 with almost the same area. Table 1 shows a summary of our results compared to state of the art implementations.

2 OCB Multi-stream Architecture

2.1 OCB Mode Description

Notation. m_i is the i^{th} plaintext message block. c_i is the i^{th} ciphertext block. AD_i is the i^{th} AD block. T_i is the tweak value related to the i^{th} block of the message or AD. K is the shared secret key. \mathcal{E}_{K,T_i} is the block cipher used by the AEAD algorithm. $\sum m_i$ is the XOR checksum of the message.

This section includes a simplified description of the OCB mode. An interested reader may refer to [4] for a full description. The OCB AEAD mode consists of two parts, shown in Figs. 2 and 3. In the original proposal [4–6], first the associated data is processed using the PMAC [19] structure shown in Fig. 3. Second, the message is encrypted using the structure in Fig. 2, computing the message checksum in parallel. Finally, the message checksum is encrypted and XOR-ed to the associated data tag to produce the final tag. These two structures, with minor changes, appear also in other encryption modes, such as OTR, SCT, CTR etc. Therefore, the ideas and techniques presented in this paper can be also beneficial for these other modes. Before describing the architecture, we present observations that inspired the architecture:

Table 1. Summary of our results compared to the state-of-the-art implementations

Algorithm	Family	Impl.	Throughput (Gbps)	Slices	Efficiency (Mbps/Slice)
AES encryption	Virtex 5	Sect. 4.1	8.0	347	23.00
		[14]	4.5	400	11.20
		[15]	46.0	3,579	12.88
	Virtex 6	Sect. 4.1	9.5	247	38.46
		[15]	64.1	3,121	20.55
AES decryption	Virtex 5	Sect. 4.1	6.1	294	<u>20.7</u>
		[14]	4.5	550	7.6
Deoxys-I-128	Virtex 6	Sect. 4.2	3.8	861	<u>4.5</u>
		[16]	2.2	946	2.57
Deoxys-I-128 encryption only	Virtex 6	Sect. 4.2	3.5	566	<u>6.2</u>
		[17]	1	920	1.12
LED	Spartan 3	Sect. 4.3	0.51	204	<u>2.5</u>
		[18]	0.19	204	0.97

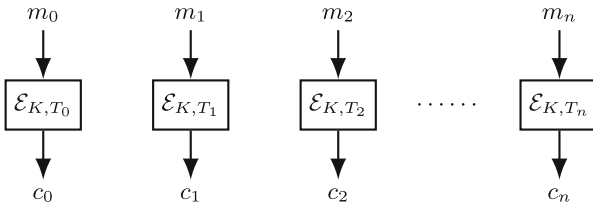


Fig. 2. The parallel encryption structure

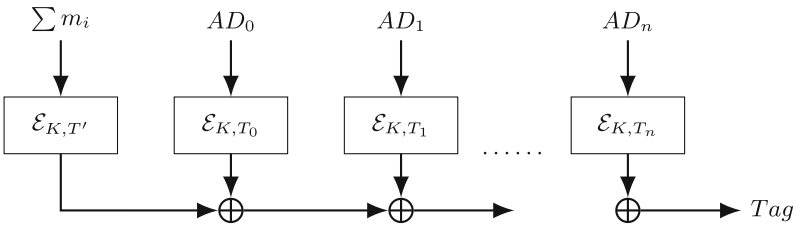


Fig. 3. The encrypt-then-xor construction

1. The first and second parts of execution do not depend on each other. Consequently, following the implementation from Poschmann and Stöttinger on the ATHENa website [2], the order can be reversed. This enables one to use the same storage for both the checksum and tag computation.
2. In Fig. 2 the computations are completely independent, while in Fig. 3, there is an output dependency between different blocks. Since there is no input dependency, both the structures are fully parallelisable. Additionally, a small temporal shift saves the temporary storage needed. For example, the first block starts at time $t = 0$ and the second block starts at $t = \Delta t$. At $t = T$ the first block is finished and stored in the tag storage. Finally, at time $t = T + \Delta t$ the second block is finished and XOR-ed with tag, in-place.

2.2 Related Work

The Cryptographic Engineering Research Group (CERG) at George Mason University (GMU), USA, runs and maintains the online platform ATHENa [16] aimed at fair, comprehensive, and automated evaluation of hardware cryptographic cores targeting FPGAs, All Programmable Systems on Chip, and ASICs. One of their on-going projects is the comparison of FPGA implementations of the CAESAR competition candidates. They have also provided high-speed round-based implementations of round 2 candidates. Among these candidates, several use OCB-like modes: OCB v1.1 [20] and AES-OTR v3.1 [21] (which use AES as underlying cipher), and Deoxys-I v1.41 [13] (which uses an AES-like tweakable block cipher Deoxys-BC). Deoxys-BC uses the same data path as AES but defines a new tweak/key-schedule that requires a smaller number of gates to evaluate when compared to AES (but with an additional 128-bit tweak input). It also requires a higher number of rounds compared to AES.

The implementations provided by the CERG team are round-based implementations that compute one cipher round per cycle. These implementations are compliant with the CAESAR Hardware API [22], developed for fair comparison among CAESAR candidates. On the other hand, a round-based implementation of Deoxys-I (encryption only) was provided by Poschmann and Stöttinger, that is not compliant with the required API. One of the requirements of the CAESAR Hardware API is to load the encryption/decryption key into the hardware core at most once per message. Since the implementation from Poschmann and Stöttinger [17] does not follow the API, it permits loading the key again with every message block, allowing the designers to get rid of the master key storage, saving 128 flip-flops. They also save 128 extra flip-flops by noticing that during the tag computation, the encryption of checksum can be computed before the associated data. This enables using the same storage for the message checksum and the intermediate tag value, saving 128 more flip flops. We follow the later approach in our implementation due to its obvious area advantage. We will see later in Sect. 4.2 that even though we target the CAESAR Hardware API compliance, our implementation of the Deoxys-I-128 encryption-only algorithm has better results compared to both the previous implementations.

2.3 Proposed Architecture

The proposed high level architecture is shown in Fig. 4. For simplicity, only the encryption data path is drawn. However, a similar data path for decryption can also be included. The architecture consists of a single round of the underlying block cipher, which is divided into N stages, each stage takes one cycle to be processed. If the block cipher requires r rounds, the architecture loads and processes N blocks, every $r \cdot N$ rounds, which leads an average latency of r cycles, equivalent to a simple single round implementation. The selection of N depends on several considerations:

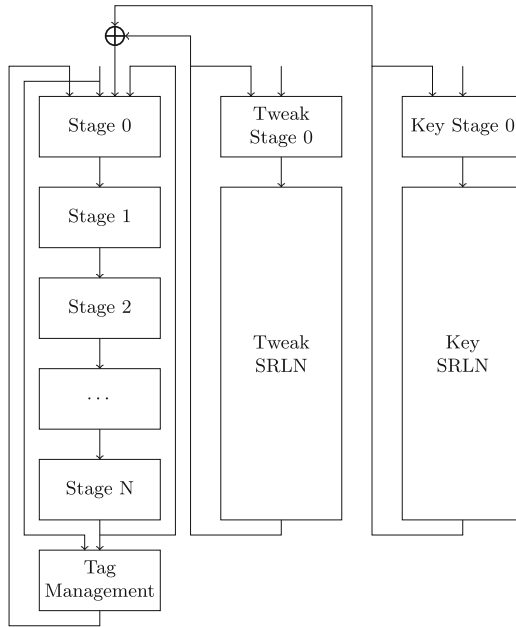


Fig. 4. Multi-stream OCB hardware architecture

1. This architecture is intended for high speed over long messages. It is noticeable that any number of blocks less than N requires the time to be encrypted. Consequently, a very large N leads to a huge overhead for short messages or for messages whose block length is not divisible by N .
2. In order to minimize the key scheduling overhead, it is performed in only one pipeline stage and then shifted N cycles. This is based on the SRL feature of the FPGA LUTs, which allows the utilization of very compact serial shift registers using logic LUTs. For most FPGAs, a single LUT can implement either a 16-bit or 32-bit SRL, which we consider as the upper bound on the value of N .

3. The pipeline registers can add a huge overhead over the simple round implementation. Therefore, in Sect. 3.3 we describe a technique to select the optimal locations of the pipeline registers in the FPGA implementation.

From these three considerations, we concluded that the optimal value for N is between 2 and 4, neglecting the control overhead. This leads to a speed-up between 2x and 4x. Additionally, for applications that require ultra high speed over very long messages, e.g. disk encryption, high speed multimedia interfaces, etc., and do not care about the area, the same architecture can be unrolled into a fully pipelined implementation. This can lead to a huge increase of the throughput. Specifically, the single round multi-stream architecture requires about $r \cdot N \cdot \lceil \frac{B}{N} \rceil$ cycles to compute B blocks. On the other hand, a fully unrolled architecture has an initial latency of $r \cdot N$ and a new block is generated every cycle, leading to a total number of cycles of $r \cdot N + B - 1$. The speed up over the round implementation is given by

$$G = \frac{r \cdot B}{r \cdot N + B - 1}$$

and for very long messages, the unrolled architecture has a speed up of r times. Since the area increases less than r times (only the round part is replicated while the tag and control part almost have the same area), the efficiency remains unchanged. In Sect. 4.1 we show that an AES round can be implemented with a clock frequency greater than 700 MHz on FPGA, with almost the same number of slices/LUTs. Therefore, we estimate that this variant can be suitable for applications that require very high speed authenticated encryption.

3 Multi-stream AES-like Ciphers

3.1 AES Data Path State-of-the-Art FPGA Implementations

AES [23] is a 128-bit block cipher, standardized in 2001 by NIST. It is based on the Substitution-Permutation Network (SPN). The internal state of the cipher can be viewed as a 4×4 matrix of bytes. It consists of 10 SPN rounds. Each round includes a `SubBytes` operation for the non-linear part, `ShiftRows` and `MixColumns` for the linear permutation and `AddRoundKey` for key addition. `SubBytes` consists of 16 independent 8-bit Sboxes, `ShiftRows` shifts the bytes in each row, independently, and `MixColumns` applies a diffusion matrix to each column, independently. All byte operations are done in $\text{GF}(2^8)$.

In this section, we quickly review state-of-the-art high speed AES-128 FPGA implementations (we only discuss full width round-based and unrolled implementations). A detailed survey on AES data paths for FPGA is provided in [12]. Full-width FPGA implementations of AES are either unrolled implementations [15], round-based single stream [24,25] or round-based multi-stream [14]. Although the scope of this paper is round-based multi-stream implementations, the optimizations described in this section can be used for any of the aforementioned

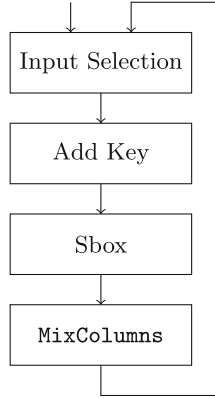


Fig. 5. The AES encryption data path from [14]

implementations. In [14], the authors proposed the AES data path shown in Fig. 5. Each box in Fig. 5 represents a pipeline stage, and it can be noticed that the selection of the pipeline stages is based on the functionality of each stage, which leads to two very fast stages in the beginning, then two slow stages afterwards. This limits the maximum possible frequency. In the next sections, we will show why this architecture might not be optimal and describe a new four-stream data path designed for FPGA to achieve higher performance efficiency.

3.2 LUT-Based Optimization of Linear Transformations

Notation. a, b, c and d are the four bytes that compose one column of the AES state. a_i is the i^{th} bit of a , where a_0 is the least significant bit. Upper-case letters A, B, C, D, E, F are the hexadecimal representations of the decimal values 10, 11, 12, 13, 14 and 15, respectively. \cdot and \oplus are multiplication and addition over $\text{GF}(2^8)$.

The AES `MixColumns` circuit is a matrix multiplication operation of the AES state byte matrix by a constant matrix M given by

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

which is a circulant MDS matrix. For AES 128-bit architectures, the `MixColumns` operation can be viewed as 16 dot-products of the vector $[2 \ 3 \ 1 \ 1]$ and a vector composed by a permutation of 4 state words. It can also be viewed as four 32-bit to 32-bit mappings (four matrix-vector products over state vectors). The later view is favorable for ASIC implementations, as it allows reducing the required number of gates by sharing many intermediate results of the computation. Specifically, only 108 XOR gates are required for implementing the 32-bit mapping [26].

However, as discussed earlier, since modern FPGAs use big 6/5 input LUTs to implement logic circuits, having a lot of small shared 2/3-input gates is not the most efficient solution. Synthesizing the circuit used in [26] or [27] for Virtex-6 FPGA requires 41 LUTs for low area and 44 LUTs for high speed. On the other hand, the dot-product view is given by

$$p = 2 \cdot a \oplus 3 \cdot b \oplus c \oplus d$$

which can be decomposed into

$$\begin{bmatrix} a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 \\ 0 & 0 & 0 & a_7 & a_7 & 0 & a_7 & a_7 \\ b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & 0 \\ 0 & 0 & 0 & b_7 & b_7 & 0 & b_7 & b_7 \\ b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\ d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \end{bmatrix} \quad (1)$$

where the elements of each column represent the inputs of one output function. From this perspective, it can be seen that 5 outputs can be implemented using one 5-input LUT, while 3 outputs can be implemented using 7-input LUT, which can be implemented using two 6-input LUTs. That sums to a total of 11 LUTs per output coefficient, 44 LUTs per output column. This shows that logic optimization does not offer much gain over the straightforward implementation of the transformation. Besides, a deeper look at the view given by the decomposition in (1) shows that the three outputs that need 7-input LUTs share two inputs bits, namely a_7 & b_7 . Decomposition (1) can be written as decomposition (2), where $x = a_7 \oplus b_7$. This decomposition can be implemented using eight 6-input LUTs and one 2-input LUT, a total of 9 LUTs per output coefficient, 36 LUTs per output column (which is smaller than the best-reported implementations) or 1.125 LUTs per output bit. It is worth mentioning that this number is near-optimal for any linear transformation over 32 bits, as the optimal number is 1 LUT/bit, which corresponds to transformation where each output bit depends on n bits, where $2 \leq n \leq 6$ (the case where $n = 1$ corresponds to an identity function and can be neglected, w.l.o.g.)¹.

$$\begin{bmatrix} a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & a_7 \\ b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & b_7 \\ 0 & 0 & 0 & x & x & 0 & x & 0 \\ b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\ d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \end{bmatrix} \quad (2)$$

¹ In fact, each 6:1 LUT can be implemented as a 5:2 LUT with shared inputs. Using this feature, our circuit can be indeed implemented using only 8 LUTs, which is the optimal figure. However, in this paper we are handling the optimization at the front-end stage and this feature is incorporated automatically by the placement and routing tool.

The optimization of the AES inverse `MixColumns` circuit is less straightforward, as M^{-1} includes larger coefficients. M^{-1} is given by

$$\begin{bmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{bmatrix}$$

A lot of work has been done on how to reuse the same circuit from M to implement M^{-1} with minimal overhead. This is done by using any of the following relations $M^{-1} = M^3$, $M^{-1} = M \cdot N$ or $M^{-1} = M \oplus K$, where N and K are matrices with low coefficients. In that direction, the circuit given by decomposition (2) will also be the smallest and the same reasoning can be used to achieve small area for both K and N . However, this approach is most useful for low area serial implementations with shared encryption/decryption data path. They do not achieve the best results for high speed round implementations with dedicated decryption data path. For example, using $M^{-1} = M^3$ requires 3.375 LUTs/bit and produces a large-depth circuit (low performance), while using $M^{-1} = M \oplus K$ is even larger. The most promising approach is $M^{-1} = M \cdot N$ which requires 288 LUTs/block, corresponding to 2.25 LUTs/bit, which is still far from optimal. On the other hand, the straightforward implementation of M^{-1} leads to output functions that include 19 input bits, which can lead to very low performance. Here, we give a circuit that requires 60 LUTs per output column, corresponding to 1.875 LUTs/bit. First, we use the same dot product view mentioned earlier, which is given by Eq. (3).

$$p' = E \cdot a \oplus B \cdot b \oplus D \cdot c \oplus 9 \cdot d = F \cdot (a \oplus b \oplus c \oplus d) \oplus (a \oplus 4 \cdot b \oplus 2 \cdot c \oplus 4 \cdot d \oplus 2 \cdot d) \quad (3)$$

Second, two observations are made

1. $F \cdot (a \oplus b \oplus c \oplus d)$ is constant across any output column.
2. $4 \cdot (a \oplus c)$ is shared by two output coefficients. The same is valid for $4 \cdot (b \oplus d)$.

Using these two observations, a circuit that requires only 60 LUTs per output column can be implemented. The circuit diagram is given in Fig. 6. This is 17% smaller than the best reported implementation. Given that `MixColumns` is the main difference between the AES encryption and decryption data paths, optimizing this primitive is crucial. On the other hand, since 1.875 LUTs/bit is still far from the optimal 1 LUT/bit figure, there may be some room for further optimization.

3.3 Zero Area Overhead Pipelining

Pipelining has been used by hardware designers/architects as a tool to increase throughput/run-time performance for a long time. However, a fully pipelined block cipher implementations can be costly, due to the large area requirements. A more realistic approach is to use multi-stream implementations. These implementations start from a sequential implementation that processes one block in

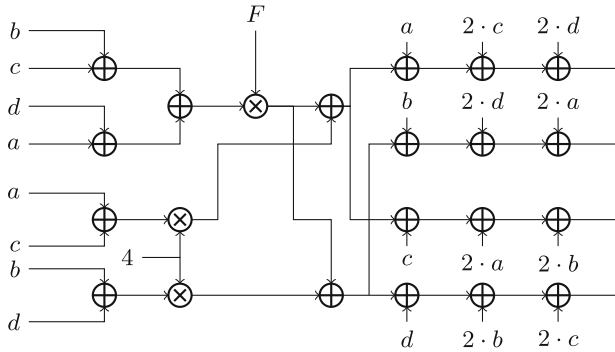


Fig. 6. FPGA-friendly inverse MixColumns circuit

C cycles, and divides it into N pipeline stages. This leads to computing x blocks in $N \cdot C$ cycles, where $x \in \{1, 2, \dots, N\}$. x depends on the number of independent block streams the user can leverage. However, this is a double-edged weapon, due to the following reasons:

1. The time required to process one block in a sequential implementation is $\sim C \cdot T$, where T is the critical path delay of the implementation. If the N pipeline stages divide the critical path evenly into segments of $\frac{T}{N}$ delay, the time required to process N blocks becomes $T + t$, where t is a small overhead, leading to $\sim Nx$ speed-up. Unfortunately, the critical path is usually not evenly divided, leading to a sub-optimal speed-up ($< N$).
2. Modern FPGA families consist of a basic building block called LUT6, which is a 6-input single-output look-up table. Additionally, each unit of this building block has an associated Flip-Flop, which the designer/synthesis tool can choose to either use it or not. In Fig. 7, we show the optimal utilization of a LUT6 unit in a pipelined architecture, where it is used to implement a 6-input circuit followed by storing the output. On the other hand, in Fig. 8, a poor selection of the location of pipeline stage is in-place, leading to the utilization of 3 look-up tables, instead of 1 in the case of Fig. 7. In other words, the poor choice of where to place the pipeline registers leads to a significant increase in area.

While the impact of moving the flip-flops 1 logic level forward in the previous example is obvious, the designers usually do not have an accurate estimation of the exact LUT utilization before synthesis. Consequently, the designers choose the pipeline stages based on the logical functions, e.g. Sbox, MixColumns, input selection, etc. In our work, we follow a different approach. First, we synthesize a single stream sequential implementation of the required block cipher. Second, we study the output layout to determine the precise distribution of pipeline stages without affecting the structure of the utilized LUTs.²

² The term zero overhead refers to the number of LUT-FF pairs, as this is the important metric, not the number of LUTs or FFs.

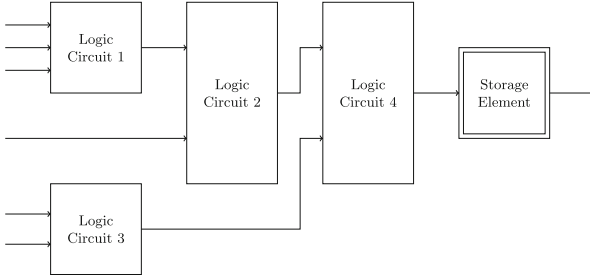


Fig. 7. Optimal pipeline selection

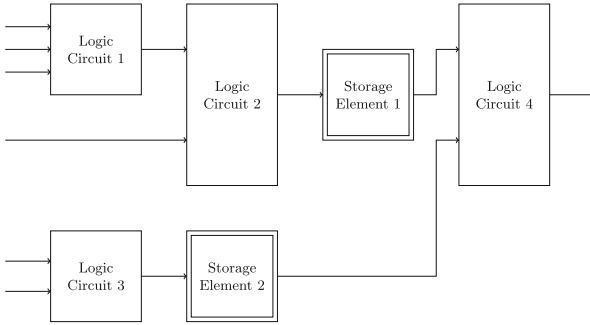


Fig. 8. Sub-optimal pipeline selection

4 Implementations and Results

4.1 Two-Stream and Four-Stream AES Implementations

Using the techniques described in Sects. 3.2 and 3.3, we have implemented two multi-stream AES data path (two and four streams), shown in Fig. 9. In addition to the use of the low area MixColumns circuit and ROM-based Sbox, the locations of the pipelining registers have been selected specifically to ensure as efficient LUT utilization as possible. In other words, both the two-stream and four-stream implementations use the same number of logical LUTs (944 LUTs, without key scheduling), out of which 95% (896 LUTs) are 6-input LUTs. The results are given in Table 2. The comparison is restricted to full-width implementations that do not utilize any BRAMs. For that reason, the implementation in [28] is not included. While it has a very high efficiency (yet smaller than ours, ~ 30 Mbps/slice), this number is biased due to the use of BRAMs to reduce the number of slices. It can be observed that our data path, with only two-streams, outperforms the data path from [14] (with four streams) in terms of both efficiency and area and reaches the same throughput. This result is achieved due to more than one factor:

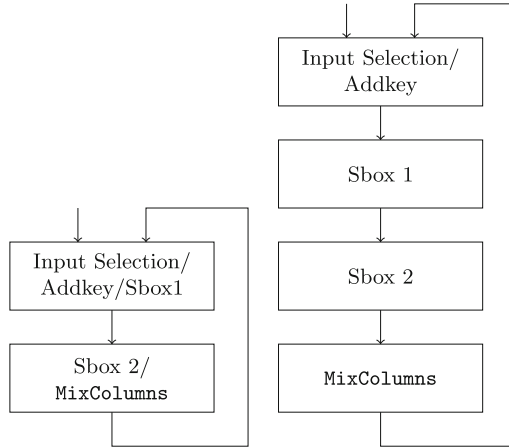


Fig. 9. Two/Four-Stream AES round data path implementation

1. The critical path in the implementation from [14] consists of three levels of logic inside the Sbox used (one LUT6 followed by MUX7 and MUX8). In our design, the critical path also consists of three levels of logic (Sbox part 2 (LUT6) and the MixColumns circuit proposed in Sect. 3.2 (LUT3 + LUT6)).
2. The MixColumns circuit used is smaller.
3. The first two pipeline stages in Fig. 5 necessitate the use of 256 LUTs. The two stages altogether can be viewed as a 6-input function, which can be easily merged into a single stage of 128 LUTs (of type LUT6).

This implies that our proposed implementation achieves the same performance as [14] for lower latency and using only independent two streams (easier to achieve). In fact, following the architecture in Sect. 2, it can be used even for slightly dependent streams (even and odd blocks of an OCB message). Additionally, by choosing to add two more stages at the output of the Sbox 2 and key addition circuits, the performance and efficiency can be further enhanced without any additional increase in the area occupied, as shown in Table 2. The results shows that our four-stream implementation outperforms all the AES FPGA implementations in the literature in terms of efficiency, to the best of our knowledge. In Table 3, we show the implementation results of the AES decryption data path. It is shown that it has a speed-up of around 2x over the similar implementation from [14].

4.2 Round-Based Two-Block Deoxys-I-128

As mentioned earlier, the Deoxys-I CAESAR candidate uses an underlying tweakable block cipher called Deoxys-BC. This cipher is similar to AES, with three major differences:

1. It consists of 14 rounds instead of 10.
2. The final round includes a MixColumns operation, as opposed to AES.

Table 2. Implementation results of AES on Virtex-5/6 FPGA (encryption only)

Family	Implementation	Key schedule	Number of slices	Max. freq. (MHz)	Throughput (Gbps)	Efficiency (Mbps/Slice)
Virtex 5	Ours/2 streams	Offline	347	350	4.5	12.90
	Ours/4 streams	Offline	347	625	8.0	23.00
	[14]/4 streams	Offline	400	350	4.5	11.20
	[15]/unrolled	Offline	3579	360	46.0	12.88
Virtex 6	Ours/4 streams	Offline	347	752	9.6	27.66
	Ours/4 streams	No	247	752	9.5	<u>38.46</u>
	[15]/unrolled	Offline	3121	501	64.1	20.55

Table 3. Implementation results of AES on Virtex-5 FPGA (decryption only)

Implementation	Key schedule	Number of slices	Max. freq. (MHz)	Throughput (Gbps)	Efficiency (Mbps/Slice)
Ours/4 streams	No	294	477	6.1	<u>20.7</u>
Ours/4 streams	Offline	445	477	6.1	13.7
[14]/4 streams	Offline	550	350	4.5	7.6

- It uses a different key schedule, which is smaller than the AES key schedule, but uses an extra public tweak value.

Based on the architecture proposed in Sect. 2 and the AES data paths proposed in Sect. 4.1, we have implemented two complete data paths for Deoxys-I. They include two and four pipeline stages, respectively. They also consist of four parts: the encryption data path, the decryption data path, the key schedule and the tweak schedule. Using the pipeline selection technique, both implementations consume the same area, except for the decryption data path which we implement only for 4 streams. In Table 4, it is shown that the bottleneck of the design, not considering the control overhead, is the decryption data path.

Table 4. Results of the Deoxys-I-128 data path implementation on Virtex-6 FPGA

Block	Number of LUT-FF pairs	Max. freq. (2-stream)	Max. freq. (4-stream)
Enc. data path	1455	492 MHz	785 MHz
Dec. data path	1170	-	665 MHz
Key schedule	442	724 MHz	724 MHz
Tweak schedule	413	620 MHz	620 MHz

Based on this datapath, a full implementation of Deoxys-I-128 has been implemented on Xilinx Virtex-6 FPGA (xc6vlx75tff784-3). The overall utiliza-

tion is ~ 3800 LUT-FF pairs and maximum operating frequency is 454 MHz. This 27% performance degradation from the datapath estimated clock frequency comes from two reasons:

1. While the datapath is very fast, there is still optimization required to the control unit to cope with such speed.
2. Although we choose a small FPGA from the Virtex 6 family, the design is still small compared to the FPGA size, which leads to it becoming I/O dominated. leading to a lot of wiring delays related to the I/O pins. This will not be applicable if the design is used as a part of a larger on-chip system.

To verify the second problem as part of the reason for performance degradation, we also implemented the design for the small Spartan-6 (xc6slx9ftg256-3) FPGA. The maximum operating frequency is 273 MHz vs. 333 MHz pre-layout (only 18% degradation). The results of the Virtex-6 implementation are summarized in Table 5. For fair comparison, we have also downloaded and implemented the *Deoxys-I-128* implementation reported on the ATHENA website [2] by CERG team. We only compare the cipher circuits without the overhead of the hardware API. Our results show an efficiency gain of 75% (1.75x) for Virtex 6 and 74% (1.74x) for Spartan-6. Table 6 shows the results for the encryption-only implementation. Our implementation is 5.536x more efficient than the implementation by (Poschmann and Stöttinger).

Table 5. Post-layout results of the *Deoxys-I-128* implementation on FPGA

FPGA	Impl.	Number of slices	Max. freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/Slice)
Virtex 6	Ours	861	454	3,874	4.5
	[16]	946	285	2,432	2.57
Spartan 6	Ours	1,010	273	2,329	2.31
	[16]	1,032	161	1,373	1.33

Table 6. Post-layout results of the *Deoxys-I-128* encryption only implementation on Virtex 6 FPGA

Impl.	Number of slices	Max. freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/Slice)
Ours	566	416	3,549	6.2
[17]	920	161	1.030	1.12

Relevance to the CAESAR Competition. The goal of this section is not to show that Deoxys-I is faster than other CAESAR candidates. The goal is to show that parallelism can actually increase the efficiency of parallelisable ciphers by significant factors. Since we are comparing two different architectures and implementations of the same cipher, it makes more sense to focus the comparison on the cipher itself. While the API provides a fair methodology for comparing different ciphers, it can hide the potential of enhancement for a certain cipher. This is exactly true for FPGA in our case, as a custom control unit that is both compliant with the CAESAR API and the 4-stream architecture has to be designed. This leads to 50% decrease in the performance of the proposed architecture during the CAESAR Competition Round 3 benchmarking stage. We emphasize that this is not the goal of the proposed architecture, as in real-life applications faster APIs can be designed, such as the one we used in the previous comparison.

Extending the Results to Other Technologies (ASIC). Since it can be argued that the techniques and optimizations in this paper are limited because they are limited to a certain technology (Xilinx FPGA), we have synthesized both the proposed implementation and the implementation from [16] for ASIC using Synopsys Design Compiler and the TSMC 65 nm technology. The results show that even with full API compliance, the proposed implementation has 54% higher throughput and is 38% more efficient. These results are summarized in Table 7. We are currently in the process of preparing the HDL code to be provided publicly soon so that other researchers can verify our results.

Table 7. Synthesis results of the Deoxys-I-128 implementation using TSMC 65 nm technology

Impl.	Area (KGE)	Max. freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/KGE)
Ours	59.53	847	7,227	121.40
[16]	53.37	549	4,684	87.76

4.3 Three-Stream LED Implementation

LED [29] is a 64-bit block cipher based on an AES-like SPN. Its state is a 4×4 matrix of 4-bit nibbles. In this paper we focus on the 64-bit key version LED-64. However, the same results can extend to the other variants of LED, since the only difference is the key scheduling part, which can be easily adjusted for this architecture. The (χ^4) round implementation from [18], Sect. 3.1, has been replicated for Spartan-3 Xilinx FPGA. Using the guidelines from Sect. 3.3, we have been able to add two extra pipeline stages at the outputs of the Sbox and the MixColumns operations, as shown in Fig. 10. In Table 8, it is shown that

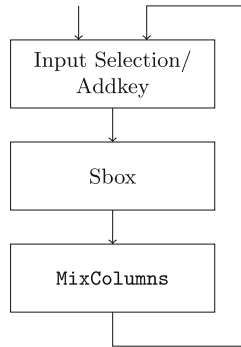


Fig. 10. Three-Stream (χ^4) LED-64 round implementation

Table 8. Results of the three-stream LED-64 implementation compared to the single stream counterpart on Spartan 3 FPGA.

Implementation	Number of slices	Number of FFs	Max. freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/Slice)
[18]/1 stream	204	74	98.7	197.35	0.97
Ours/3 streams	204	202	257	514	<u>2.5</u>

almost all the available flip-flops has been used, increasing both the throughput and efficiency by 2.57x at no additional area cost.

Acknowledgments. The authors would like to thank the anonymous referees for their helpful comments. This work is partly supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. CAESAR Competition: CAESAR submissions (2016). <https://competitions.cr.yt.caesar-submissions.html>
2. George Mason University: ATHENA: Automated Tools for Hardware EvaluationN (2017). <https://cryptography.gmu.edu/athena/>
3. Abed, F., Forler, C., Lucks, S.: General classification of the authenticated encryption schemes for the CAESAR competition. *Comput. Sci. Rev.* (2016)
4. Rogaway, P., Bellare, M., Black, J.: OCB: a block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **6**(3), 365–403 (2003)
5. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30539-2_2
6. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21702-9_18

7. Peyrin, T., Seurin, Y.: Counter-in-tweak: authenticated encryption modes for tweakable block ciphers. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 33–63. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_2
8. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 275–292. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_16
9. Boyar, J., Peralta, R.: A new combinational logic minimization technique with applications to cryptology. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 178–189. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13193-6_16
10. Boyar, J., Peralta, R.: A small depth-16 circuit for the AES S-box. In: Grizalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IAICT, vol. 376, pp. 287–298. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30436-1_24
11. Canright, D.: A very compact S-Box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_32
12. Resende, J.C., Chaves, R.: AES datapaths on FPGAs: a state of the art analysis. In: Sklavos, N., Chaves, R., Di Natale, G., Regazzoni, F. (eds.) Hardware Security and Trust, pp. 1–25. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-44318-8_1
13. Jean, J., Nikolic, I., Peyrin, T., Seurin, Y.: Deoxys v1.41. Technical report, Nanyang Technological University, Singapore/ANSSI, Paris, France (2016)
14. Bulens, P., Standaert, F.-X., Quisquater, J.-J., Pellegrin, P., Rouvroy, G.: Implementation of the AES-128 on Virtex-5 FPGAs. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 16–26. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68164-9_2
15. Liu, Q., Xu, Z., Yuan, Y.: A 66.1 GBPS single-pipeline AES on FPGA. In: 2013 International Conference on Field-Programmable Technology (FPT), pp. 378–381, December 2013
16. Deoxys-I-128 implementation by cerg team (2016). <https://cryptography.gmu.edu/athena/>
17. Poschmann, A., Stöttinger, M.: Deoxys-I-128 implementation by poschmann and Stöttinger (2016). <https://cryptography.gmu.edu/athena/>
18. Nalla Anandakumar, N., Peyrin, T., Poschmann, A.: A very compact FPGA implementation of LED and PHOTON. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 304–321. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13039-2_18
19. Black, J., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 384–397. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_25
20. Krovetz, T., Rogaway, P.: Ocb (v1. 1) (2016)
21. Minematsu, K.: AES-OTR v3.1. Technical report, NEC Corporation, Japan (2016)
22. Homsirikamol, E., Diehl, W., Ferozpur, A., Farahmand, F., Yalla, P., Kaps, J.P., Gaj, K.: CAESAR Hardware API. Cryptology ePrint Archive, Report 2016/626 (2016)
23. NIST: National Institute of Standards and Technology: Advanced Encryption Standard AES (2001)
24. El Maraghy, M., Hesham, S., El Ghany, M.A.A.: Real-time efficient FPGA implementation of AES algorithm. In: 2013 IEEE 26th International SOC Conference (SOCC), pp. 203–208. IEEE (2013)

25. Chaves, R., Kuzmanov, G., Vassiliadis, S., Sousa, L.: Reconfigurable memory based AES co-processor. In: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, 8-pp. IEEE (2006)
26. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-AES v 2.0. Cryptology ePrint Archive, Report 2016/1005 (2016)
27. Ghaznavi, S., Gebotys, C., Elbaz, R.: Efficient technique for the FPGA implementation of the AES mixcolumns transformation. In: International Conference on Reconfigurable Computing and FPGAs, ReConFig 2009, pp. 219–224. IEEE (2009)
28. Resende, J.C., Chaves, R.: Compact dual block AES core on FPGA for CCM protocol. In: 2015 25th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–8. IEEE (2015)
29. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23951-9_22

Improved Differential Cryptanalysis on Generalized Feistel Schemes

Ivan Tjuawinata^(✉), Tao Huang, and Hongjun Wu

Division of Mathematical Sciences, School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore, Singapore
S120015@e.ntu.edu.sg, {huangtao,wuhj}@ntu.edu.sg

Abstract. Nachev *et al.* used differential cryptanalysis to study four types of Generalized Feistel Scheme (GFS). They gave the lower bound of maximum number of rounds that is indistinguishable from a random permutation. In this paper, we study the security of several types of GFS by exploiting the asymmetric property. We show that better lower bounds can be achieved for the Type-1 GFS, Type-3 GFS and Alternating Feistel Scheme. Furthermore, we give the first general results regarding to the lower bound of the Unbalanced Feistel Scheme.

Keywords: Generalized Feistel Network · Differential analysis
Chosen ciphertext attack · Known Plaintext Attack

1 Introduction

1.1 Background

The Feistel Network is a widely-used method used to construct iterated block ciphers. It has similar operations in encryption and decryption process which is hardware efficient and the round function is not required to be a bijective function. It has been applied in many block ciphers such as DES, DEAL [9] and Camellia [2]. The structure was generalized to allow more branches and different relations between the branches to form Generalized Feistel Network (GFN) [13]. Before the term GFN was proposed, Zheng *et al.* [24] described 3 types of transformations which were in fact Type-1, Type-2 and Type-3 Generalized Feistel Schemes. Anderson and Biham [1] and Lucks [11] proposed block cipher designs using Alternating Feistel Network. Another type of GFN is the Unbalanced Feistel Scheme, which was designed by Schneier and Kelsey [19]. Many block cipher designs employed the GFN, such as CLEFIA [20], Skipjack and Simpira [6]. The advantage of using a Generalized Feistel Network is that it allows for a design to handle a larger block size with a relatively small round function.

1.2 Previous Work

Many analysis on Feistel network and Generalized Feistel Network have been done [7, 10, 12, 14, 15, 22]. However, as mentioned in [7], most analysis is specialized in some types instead of analysing many types at once.

Nachev *et al.* [12] used differential cryptanalysis to study four types of GFS using Known Plaintext Attack (KPA) and Chosen Plaintext Attack (CPA) model. They established lower bounds of the maximum number of rounds distinguishable in Type-1, Type-2, Type-3 and Alternating Feistel Scheme in the two models.

Provable-security analysis has been applied to Feistel Networks in [7, 10]. Luby and Rackoff [10] analysed the classical Feistel Networks which is then improved and generalised by Hoang and Rogaway in [7] to analyse Classical, Unbalanced, Alternating, Type-1, Type-2 and Type-3 Generalized Feistel Scheme. The theoretical analysis of Generalized Feistel also plays an important role in design and analysis of practical ciphers. In the design of DEAL [9], Knudsen considers this theoretical attack to provide a security bound for any key schedule that is used.

An interesting property existed in many the GFS designs is that the encryption and decryption are not exactly the same, which sometimes makes the differential propagation slower in the decryption than in the encryption. In the analysis on Skipjack [4, 5], the difference in the decryption has been considered. Recently, Tjuawinata *et al.* [21] showed that the analysis of Simpira [6] can be improved by considering the asymmetry of Type-1 Generalized Feistel Scheme. While this property is exploited in cryptanalysis, it is undesired for the designer. In the design criterion of Keccak [3], it mentioned the property that the same permutation function is used in both encryption and decryption.

1.3 Our Contribution

In this paper, we study the asymmetric property in the Generalized Feistel Schemes. We provide better lower bounds of the maximum number of rounds distinguishable in 3 different types of Generalized Feistel Networks given in [12], which are Type-1 Feistel Scheme, Type-3 Feistel Scheme and Alternating Feistel Scheme.¹ We also provide a lower bound of the maximum number of rounds distinguishable in another type of Generalized Feistel Network, the Unbalanced Feistel Network. As far as we know, this is the first result on Unbalanced Feistel Network that is applicable to different values of k' . We exploit the asymmetry of certain types of GFS by observing that the backward differential diffusion is slower than the forward differential diffusion. This leads to the improvements on the lower bounds.

For Type-1 Feistel Scheme, we provide a chosen ciphertext distinguisher which distinguishes $k - 1$ more rounds than the distinguisher given in [12] with the same complexity. Furthermore, when the number of rounds to distinguish is fixed to $ak - 2$ rounds for some integer a in the range $4 \leq a \leq k - 1$, the distinguisher in this paper has complexity $1/2^n$ of the distinguisher given in the CPA model in [12], from $\sqrt{2} \cdot 2^{(a-2)n}$ to $\sqrt{2} \cdot 2^{(a-3)n}$.

¹ We also examine Type-2 Feistel Scheme, but we cannot improve the previous results since it does not have asymmetric property.

In Type-3 Feistel Scheme, [12] only provides lower bound for the case when the number of branches is at least 6. We propose a distinguisher which can be used for any number of branches and can distinguish up to $k + 2$ rounds in both KPA and CCA model with complexity $\sqrt{2} \cdot 2^{(k-1)n}$ and $\sqrt{2} \cdot 2^{(k-2)n}$ respectively. When k is at least 6, in the CCA model, a distinguisher for one more rounds than the one given in [12] is constructed.

In Alternating Feistel Scheme, our analysis shows that lower complexity can be achieved in some special cases. More specifically, when the number of rounds is odd, the complexity is improved by a factor of $2^{\frac{3n}{2}}$ from the distinguisher proposed in [12].

In our analysis of Unbalanced Feistel Scheme, let k be the total number of sub-blocks and k' be the number of sub-blocks that are used as the output of the round function. In this paper, we consider two special cases when k' or $k - k'$ divides k . When $k' = 1$, we can distinguish up to $(k^2 + k - 1)$ rounds with complexity less than 2^{kn} in the KPA model. In the CCA model, the number of rounds that can be distinguished is up to $2k$ rounds with complexity less than 2^n . When $k' \geq 1$, a lower bound of the maximum number of rounds that is distinguishable from random permutation is given. In the KPA model, the bound is $\frac{k^2}{k'} - \frac{k}{2} + \frac{k}{k'}$ when k' is even and $\frac{k^2}{k'} - \frac{k(k-1)}{2k'}$ when k' is odd. In the CCA model, the bound is $\frac{k}{2} + 2\frac{k}{k'}$ when k' is even and $\frac{k(k'+3)}{2k'}$ when k' is odd. To the best of our knowledge, this is the first analysis on Unbalanced Feistel Scheme for any values of k .

1.4 Organization

We give some preliminaries in Sect. 2. The attack overview is then discussed in Sect. 3. The analysis on Type-1 Feistel Scheme is presented in Sect. 4. Sections 5 and 6 contains analysis of Type-3 and Alternating Feistel Scheme. The Unbalanced Feistel Scheme is analysed in Sect. 7. In Sect. 8, we conclude this paper.

2 Preliminaries

2.1 Generalized Feistel Schemes

A Generalized Feistel Scheme of branch k is defined as a (keyed)-permutation $\Pi : (\mathbb{F}_{2^n})^k \rightarrow (\mathbb{F}_{2^n})^k$. For the m input-output pairs of Π , for all $i \in \{0, \dots, m - 1\}$, the i -th input and output of Π are denoted by $(I_0(i), \dots, I_{k-1}(i))$ and $(S_0(i), \dots, S_{k-1}(i))$ respectively. Since the analysis is on the inverse of Π , in the remaining of the paper, “input” refers to $(S_0(i), \dots, S_{k-1}(i))$ while “output” refers to $(I_0(i), \dots, I_{k-1}(i))$. In this paper, four types of Generalized Feistel schemes are considered in details:

Type-1 Feistel Schemes. Π is an r -round Type-1 Feistel scheme if Π consists of r repetitions of $\mu_1 : (\mathbb{F}_{2^n})^k \rightarrow (\mathbb{F}_{2^n})^k$ where $\mu_1(x_0, \dots, x_{k-1}) = (x_1 \oplus$

$F_i(x_0, x_2, \dots, x_{k-1}, x_0)$. Assume that $F_i : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$ is a function from n -bit input to n -bit output which may vary depending on the round where it is being called. Here $i = 1, \dots, r$. Illustration of round i of Type-1 Feistel Scheme can be found in Fig. 1.

Type-3 Feistel Schemes. Π is an r -round Type-3 Feistel scheme if Π consists of r iterations of $\mu_3 : \mathbb{F}_{2^n}^k \rightarrow \mathbb{F}_{2^n}^k$. Given (x_0, \dots, x_{k-1}) , μ_3 maps

$$(x_0, \dots, x_{k-1})$$

to

$$(x_1 \oplus F_{(i,0)}(x_0), x_2 \oplus F_{(i,1)}(x_1), \dots, x_{k-1} \oplus F_{(i,k-2)}(x_{k-2}), x_0).$$

Figure 2 illustrates the i -th round of Type-3 Feistel Scheme.

Alternating Feistel Schemes. For this scheme, consider two different round functions $\mu_{A,0}, \mu_{A,1} : \mathbb{F}_{2^n}^k \rightarrow \mathbb{F}_{2^n}^k$ which are used alternately for each round.

- $\mu_{A,0}(x_0, \dots, x_{k-1}) = (x_0 \oplus F_i(x_1, \dots, x_{k-1}), x_1, \dots, x_{k-1})$ where $F_i : \mathbb{F}_{2^n}^{k-1} \rightarrow \mathbb{F}_{2^n}$ is called in round $2i - 1$. $\mu_{A,0}$ is called the contracting round.
- $\mu_{A,1}(x_0, \dots, x_{k-1}) = (x_0, x_1 \oplus F_{(i,1)}(x_0), \dots, x_{k-1} \oplus F_{(i,k-1)}(x_0))$. Here $F_{i,j} : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$ is the function called in the j -th component in round $2i$.

These rounds are called the expanding rounds.

Illustration of round $2i - 1$ and $2i$ of Alternating Feistel Scheme can be found in Fig. 3. Note that round number and index i starts from 1 instead of 0. Alternatively, $\mu_{A,1}$ can be used in odd rounds and $\mu_{A,0}$ in even rounds but in this paper a contracting round is always used at round 1. Note that if $\mu_{A,1}$ is used in the first one instead, the backward analysis on this variant is equivalent to the forward analysis discussed in [12].

Unbalanced Feistel Schemes. This is a special case of the UFN defined in Fig. 1 of [7]. Let $k' = 1, \dots, k - 1$ and $F_s : \mathbb{F}_{2^n}^{k-k'} \rightarrow \mathbb{F}_{2^n}^{k'}$ be a map from $(k - k')n$ bit to $k'n$ bit with component functions denoted as $F_{s,0}, \dots, F_{s,k'-1}$ with the round number s as its parameter. Then Π is an r -round UFN(k', k) if it contains r repetitions of $\mu_U : \mathbb{F}_{2^n}^k \rightarrow \mathbb{F}_{2^n}^k$. In round s , given an input (x_0, \dots, x_{k-1}) , μ_U maps it to

$$(x_{k'}, \dots, x_{k-1}, x_0 \oplus F_{s,0}(x_{k'}, \dots, x_{k-1}), \dots, x_{k'-1} \oplus F_{s,k'-1}(x_{k'}, \dots, x_{k-1})).$$

Figure 4 provides an illustration of round s of UFN(k', k).

In this paper, differential analysis on the inverse of Π is considered. So the attack starts with the image $(S_0(i), \dots, S_{k-1}(i))$ and the differential path is built to the preimage, $(I_0(i), \dots, I_{k-1}(i))$.

2.2 Random Variable

Given a random variable X , denote by $E(X), V(X), \sigma(X)$ the expected value, variance and standard deviation of X respectively. Note that $V(X) = E(X^2) - E(X)^2$ and $\sigma(X) = \sqrt{V(X)}$.

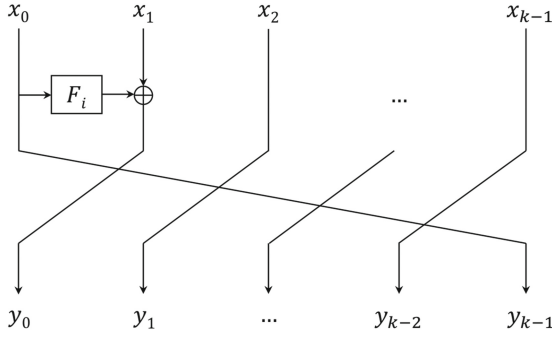


Fig. 1. Round i of Type-1 Feistel Scheme

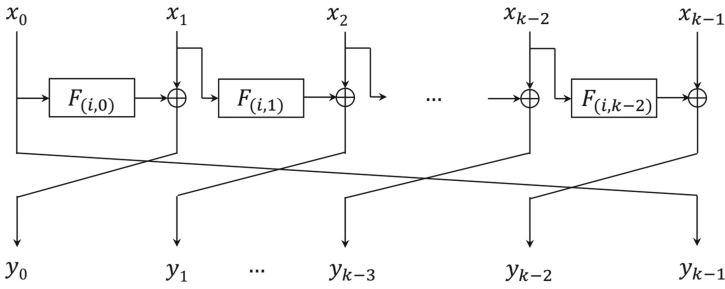


Fig. 2. Round i of Type-3 Feistel Scheme

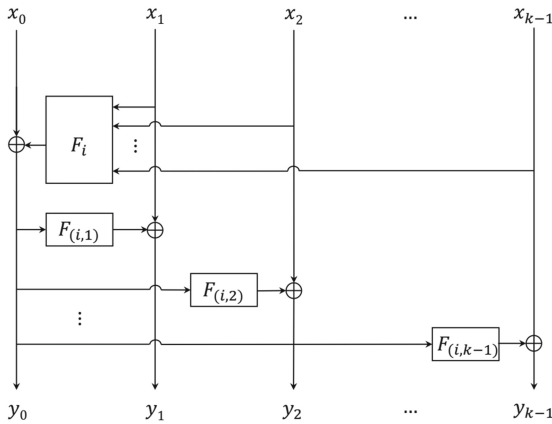


Fig. 3. Round $2i - 1$ and Round $2i$ of Alternating Feistel Scheme

Now given n random variables X_1, \dots, X_n , define the covariance of X_i and X_j as $Cov(X_i, X_j) = E(X_i X_j) - E(X_i)E(X_j)$. A simple calculation of the definition yields $V(\sum_{i=1}^n X_i) = \sum_{i=1}^n V(X_i) + \sum_{i \neq j, 1 \leq i, j \leq n} Cov(X_i, X_j)$.

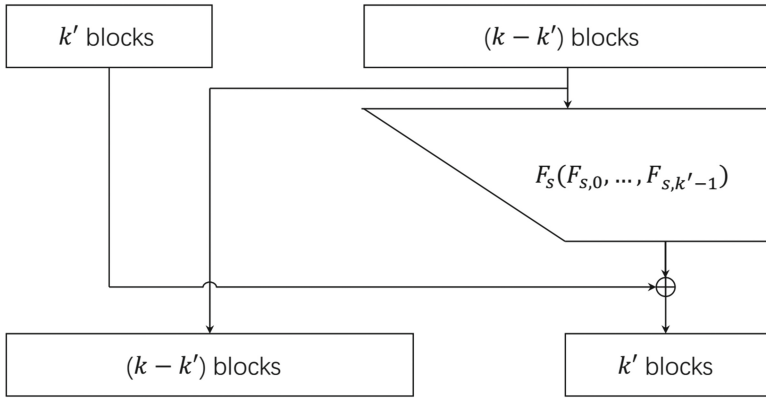


Fig. 4. Round s of Unbalanced Feistel Scheme

Proposition 1 [12]. Let X and Y be two random variables. X is said to be distinguishable from Y if $|E(X) - E(Y)| \geq \max(\sigma(X), \sigma(Y))$.

More specifically, let E_X, E_Y be the expected values of X and Y respectively while σ_X, σ_Y being the standard deviations of X and Y respectively. Without loss of generality, let $E_X < E_Y$. Then, if $E_Y - E_X \geq \max(\sigma(X), \sigma(Y))$:

1. $Pr \left(X \geq \frac{E_X + E_Y}{2} \right) \leq 0.30854$
2. $Pr \left(Y \leq \frac{E_X + E_Y}{2} \right) \leq 0.30854$.

Proof. We only prove the first claim since the second one can be proved by using the same method. A simple calculation tells us that:

$$\begin{aligned}
 Pr \left(X \geq \frac{E_X + E_Y}{2} \right) &= Pr \left(X - E_X \geq \frac{E_Y - E_X}{2} \right) \\
 &\leq Pr \left(X - E_X \geq \frac{\sigma_X}{2} \right) \\
 &= Pr \left(\frac{X - E_X}{\sigma_X} \geq \frac{1}{2} \right).
 \end{aligned}$$

Assuming that X is sampled large enough time, we can use the Central Limit Theorem to approximate $\frac{X - E_X}{\sigma_X}$ by a standard normal distribution. Hence by using this approximation and the standard normal distribution table, we get the upper bound claimed.

Remark 1. When we use Proposition 1, the random variables are actually the number of plaintext-ciphertext pairs that satisfy some equations. Now since the number of plaintext-ciphertext pairs is $\mathcal{O}(2^{\alpha n})$ for some constant α , we can apply Central Limit Theorem here. So if the random variable is \mathcal{X} with mean μ and standard deviation δ , we can approximate $\frac{\mathcal{X} - \mu}{\delta}$ by the standard normal distribution.

3 Attack Overview

In this paper, as we have discussed in Sect. 1.2, we exploit the asymmetry of the scheme by considering the backward differential diffusion.

We will discuss two types of improvements:

Unconstrained environment. We aim for a better lower bound for the maximum number of rounds distinguishable than the bound given in [12]. By unconstrained environment, we mean that analysis considered in this environment aims to distinguish more rounds than the previous results with complexity strictly less than 2^{kn} where k is the number of sub-block and n is the number of bits in each sub-block. Throughout this paper, the complexity of the attack is measured by the number of queries performed to make the attack possible.

Constrained environment. There are two possible forms of this improvement. Firstly, we aim to improve the number of rounds that can be distinguished in the backward direction given the same complexity as the distinguisher given in [12]. Secondly, given the same number of rounds, we aim to reduce the complexity to distinguish the GFS from a random permutation.

Our analysis uses m plaintext-ciphertext pairs and considers the expected number of pairs \mathcal{N} that satisfies certain conditions depending on the scheme analysed. Let $\mathcal{N}_{\text{perm}}$ be the value of \mathcal{N} for a random permutation and $\mathcal{N}_{\mathcal{F}}$ be the value \mathcal{N} for \mathcal{F} , the r -round Generalized Feistel Scheme. We use this information to calculate the maximum number of rounds such that $\mathcal{N}_{\text{perm}}$ is distinguishable from $\mathcal{N}_{\mathcal{F}}$.

The functions F_i (or $F_{(i,j)}$) used in the round function of GFS are assumed to be ideal keyed functions. Given the input, the output is a random n -bit string. Similarly, since it is ideal, given a nonzero input difference, the output difference is uniformly distributed.

Furthermore, let \mathfrak{J}_1 and \mathfrak{J}_2 be two distinct indices of the round function in the same cryptosystem (I_j can be a single integer or a pair or integers depending on the GFN we are considering). Given two different indices values \mathfrak{J}_1 and \mathfrak{J}_2 , we also assume that $F_{\mathfrak{J}_1}$ and $F_{\mathfrak{J}_2}$ are independent from each other. Hence given the same input (or output) difference ΔS of $F_{\mathfrak{J}_1}$ and $F_{\mathfrak{J}_2}$, we can further assume that $F_{\mathfrak{J}_2}(\Delta S)$ is uniformly distributed even assuming that $F_{\mathfrak{J}_1}(\Delta S)$ is already known.

First we give an intuitive description on how to launch the attack. Suppose that given a Generalized Feistel Scheme f of \mathfrak{r} rounds, we denote the differences in each stage as $\Delta_I - \Delta_1 - \dots - \Delta_{\mathfrak{r}-1} - \Delta_O$. The first step of the attack is done by expressing Δ_I as a function of $\Delta_1, \dots, \Delta_{\mathfrak{r}-1}$ and Δ_O . We are choosing the express Δ_I as a function of Δ_O instead of the other way around since we want to use the expression we get to launch a backward differential trail instead of the forward trail. To enable this, for each Δ , we partition Δ into k sub-blocks. Since each round function takes one of these sub-blocks as input, we can easily find the expression that we need.

Having these expressions, we can then choose carefully the input and output difference (truncated differences) to maximize the probability for the specified input difference focusing in some of the sub-blocks to lead to the output difference chosen. To calculate the success probability, we consider the number of ciphertext pairs with specified difference that can lead to the plaintext pairs with the chosen input difference in two scenarios; when the function is a random permutation, denoted by $\mathcal{N}_{\text{perm}}$ and when the function is in the form of the Generalized Feistel Network considered, denoted by $\mathcal{N}_{\mathcal{F}}$.

Now having the expected values and variances of both $\mathcal{N}_{\text{perm}}$ and $\mathcal{N}_{\mathcal{F}}$, those four values will be functions of the round number r and number of ciphertext pairs with the chosen difference m . By Definition 1, $\mathcal{N}_{\mathcal{F}}$ is distinguishable from $\mathcal{N}_{\text{perm}}$ if $|E(\mathcal{N}_{\mathcal{F}}) - E(\mathcal{N}_{\text{perm}})| \geq \max(\sigma(\mathcal{N}_{\mathcal{F}}), \sigma(\mathcal{N}_{\text{perm}}))$. So using this inequality, we obtain a relation between the number of rounds r and the number of ciphertext pairs m . This will give us a lower bound of m given r . Since we want the distinguisher to be useful, we require m to be less than the total number of possible ciphertext pairs. In the case of known ciphertext attack, this means that we need $m \leq 2^{kn}$. This gives us an upper bound for the round number, r , such that \mathcal{F} is distinguishable from a random permutation using this backward differential attack.

As we described above, in fact the main idea of the attack is exactly the same for all the types of Generalized Feistel Scheme. We first calculate a relation between ciphertext and plaintext differences which is closely related to the structure of the scheme. Once the relation is established, the calculation of the expectation and standard deviation will be very similar and they will be independent of the scheme. Because of this similarity, we will just describe the calculation once and omit the others. In the following sections of this paper, we perform this attack on different types of Generalized Feistel Networks discussed in Sect. 2.1.

4 Type-1 Feistel Schemes

4.1 Analysis of the Type-1 Feistel Schemes

For this analysis, we assume the number of rounds is $r = ak + b$ where k is the number of branches in the scheme and a and b are non-negative integers where $0 \leq b \leq k - 1$ and $k \geq 3$. We will be using the notation described in the previous section for our analysis, namely I_0, \dots, I_{k-1} for the k sub-blocks pre-image of Π while S_0, \dots, S_{k-1} is used to denote the k sub-blocks image of Π . In this section we discuss in detail how we build the relations between the sub-blocks, then we discuss how we choose the differential trail. Having the differential trail, the expected value and the variance of the trail when Π is random permutation and a type-1 Generalized Feistel Scheme are calculated. This in turns tells us the maximum number of rounds that is distinguishable from a random permutation using the chosen differential.

Let X_i be the intermediate variables obtained in the second branch (indexed as 1) after the i -th round in the backward direction. By definition of the round function of Type-1 Feistel Scheme, we have the following relations:

$$\begin{aligned} X_0 &= S_{k-1}, \\ X_1 &= S_0 \oplus F_{ak+b}(S_{k-1}), \\ \text{For } t &= 2, \dots, k-1, X_t = S_{k+1-t} \oplus F_{ak+b-(t-1)}(S_{k-t}), \\ \text{For } t \geq k, X_t &= X_{t-k} \oplus F_{ak+b-(t-1)}(X_{t-(k-1)}), \text{ Note that } F_r \text{ is always used} \\ &\text{with input } X_{ak+b-r-k+2}. \\ \text{For } r \geq k-1, &\text{ the input of the } r\text{-th round in the backward direction is} \\ &(X_{r-(k-1)}, X_r, X_{r-1}, \dots, X_{r-(k-2)}). \end{aligned}$$

After $r = ak + b$ rounds, the state becomes (I_0, \dots, I_{k-1}) where $I_0 = X_{(ak+b)-(k-1)}$ and for $i = 1, \dots, k-1, I_i = X_{(ak+b)-(i-1)}$. The following equalities can then be derived using the relations established above:

$$\begin{aligned} I_0 &= X_{b+1} \oplus \bigoplus_{i=0}^{a-2} F_{(a-i-1)k}(X_{ik+b+2}), \\ \text{For } j \in \{1, \dots, \min(k-1, b+1)\}, \end{aligned}$$

$$I_j = X_{b+1-j} \oplus \bigoplus_{i=0}^{a-1} F_{(a-i-1)k+j}(X_{ik+(b+2-j)}),$$

For $j \in \{\min(k-1, b+1) + 1, \dots, k-1\}$,

$$I_j = X_{k+b+1-j} \oplus \bigoplus_{i=0}^{a-2} F_{(a-i-2)k+j}(X_{ik+(k+b+2-j)}).$$

In particular, for I_1 ,

$$\begin{aligned} I_1 &= X_b \oplus \bigoplus_{i=0}^{a-1} F_{(a-i-1)k+1}(X_{ik+(b+1)}) \\ &= X_b \oplus \bigoplus_{i=0}^{a-2} F_{(a-i-1)k+1}(X_{ik+(b+1)}) \oplus F_1(I_0) \\ &= \begin{cases} S_1 \oplus \bigoplus_{i=0}^{a-2} F_{(a-i-1)k+1}(X_{ik+1}) \oplus F_1(I_0) & \text{if } b = 0 \\ S_0 \oplus F_{ak+1}(S_{k-1}) \oplus \bigoplus_{i=0}^{a-2} F_{(a-i-1)k+1}(X_{ik+2}) \oplus F_1(I_0) & \text{if } b = 1 \\ S_{k+1-b} \oplus F_{ak+1}(S_{k-b}) \oplus \bigoplus_{i=0}^{a-2} F_{(a-i-1)k+1}(X_{ik+(b+1)}) \\ \oplus F_1(I_0) & \text{otherwise.} \end{cases} \end{aligned}$$

We can further expand the sum by noting that when $i = 0$, the summand is $F_{(a-i-1)k+1}(X_{b+1})$ and

$$X_{b+1} = \begin{cases} S_0 \oplus F_{ak}(S_{k-1}) & \text{if } b = 0 \\ S_{k-b} \oplus F_{ak}(S_{k-b-1}) & \text{if } 1 \leq b \leq k-2 \\ S_1 \oplus F_{ak}(S_0 \oplus F_{ak+(k-1)}(S_{k-1})) & \text{if } b = k-1. \end{cases}$$

So in any value of $b \in \{0, \dots, k-1\}$, we can express I_1 as a function of several sub-blocks of the output S_j , $F_1(I_0)$ and $a-2$ terms determined by intermediate variables. More specifically, for $b \in \{0, \dots, k-1\}$, we have:

1. When $b = 0$,

$$\begin{aligned} I_1 \oplus S_1 \oplus F_1(I_0) &= \bigoplus_{i=1}^{a-2} F_{(a-i-1)k+1}(X_{ik+1}) \oplus F_{(a-1)k+1}(S_0 \oplus F_{ak}(S_{k-1})), \end{aligned} \quad (1)$$

2. When $b = 1$,

$$\begin{aligned} I_1 \oplus S_0 \oplus F_1(I_0) &= \bigoplus_{i=1}^{a-2} F_{(a-i-1)k+1}(X_{ik+2}) \oplus F_{(a-1)k+1}(S_{k-1} \oplus F_{ak}(S_{k-2})), \end{aligned} \quad (2)$$

3. When $2 \leq b \leq k-2$,

$$\begin{aligned} I_1 \oplus S_{k+1-b} \oplus F_1(I_0) &= \bigoplus_{i=1}^{a-2} F_{(a-i-1)k+1}(X_{ik+2}) \oplus F_{(a-1)k+1}(S_{k-b} \oplus F_{ak}(S_{k-b-1})), \end{aligned} \quad (3)$$

4. When $b = k-1$,

$$\begin{aligned} I_1 \oplus S_2 \oplus F_1(I_0) &= \bigoplus_{i=1}^{a-2} F_{(a-i-1)k+1}(X_{ik+1}) \oplus F_{(a-1)k+1}(S_1 \oplus F_{ak}(S_0 \oplus F_{(a+1)k-1}(S_{k-1}))). \end{aligned} \quad (4)$$

To choose the truncated differential for each case, we try to utilize Eqs. (1), (2), (3) and (4). We will describe how we choose it for the case when $b = 0$. The same idea can then be applied to all the other cases.

Note that for this case, for any ciphertext and its plaintext, we have the relation

$$\begin{aligned} I_1 \oplus S_1 \oplus F_1(I_0) &= \bigoplus_{i=1}^{a-2} F_{(a-i-1)k+1}(X_{ik+1}) \oplus F_{(a-1)k+1}(S_0 \oplus F_{ak}(S_{k-1})), \end{aligned}$$

Now for any two ciphertexts $C = (S_0, \dots, S_{k-1})$, $C' = (S'_0, \dots, S'_{k-1})$ that we choose (and their corresponding plaintexts $P = (I_0, \dots, I_{k-1})$, $P' = (I'_0, \dots, I'_{k-1})$), we can only determine the value in the left hand side of Eq. (1). So based on this relation, we try to find the probability that $I_1 \oplus S_1 \oplus F_1(I_0) = I'_1 \oplus S'_1 \oplus F_1(I'_0)$. Since F_1 is always assumed to be ideal, after some rearrangement, this probability is the same as the probability that:

1. $I_0 = I'_0$
2. $I_1 \oplus I'_1 = S_1 \oplus S'_1$.

So we will use this as the truncated differential for the case of Type-1 Scheme with $b = 0$. As mentioned before, this is done by collecting m ciphertexts with their respective plaintexts and we compute the number of ciphertext pairs (along with their corresponding plaintexts) that satisfies the above conditions. The same analysis is done to all the other cases.

Now to find a theoretical approximation for the probability of these conditions to be satisfied in various cases, we use the fact that if $I_0 = I'_0$ and $I_1 \oplus I'_1 = S_1 \oplus S'_1$, we must have

$$\bigoplus_{i=1}^{a-2} F_{(a-i-1)k+1}(X_{ik+1}) \oplus F_{(a-1)k+1}(S_1 \oplus F_{ak}(S_0 \oplus F_{(a+1)k-1}(S_{k-1})))$$

is equal to

$$\bigoplus_{i=1}^{a-2} F_{(a-i-1)k+1}(X'_{ik+1}) \oplus F_{(a-1)k+1}(S'_1 \oplus F_{ak}(S'_0 \oplus F_{(a+1)k-1}(S'_{k-1}))).$$

Now note that in this last equation, we have terms that are just functions of S_0, S_{k-1}, S'_0 and S'_{k-1} . So in the chosen ciphertext attack, to increase the probability, we can make sure that these terms are equal in both sides by making sure that $S_0 = S'_0$ and $S_{k-1} = S'_{k-1}$. So in the chosen ciphertext attack, instead of choosing m random ciphertexts, we choose them with their first and last sub-blocks being fixed to a predetermined value.

In summary, out of the m plaintext-ciphertext pairs, we count the number of $(s, t), 1 \leq s < t \leq m$ such that

$$\begin{aligned} 1. \quad I_0(s) &= I_0(t) \\ 2. \quad I_1(s) \oplus I_1(t) &= \begin{cases} S_1(s) \oplus S_1(t) & \text{if } b = 0 \\ S_0(s) \oplus S_0(t) & \text{if } b = 1 \\ S_{k+1-b}(s) \oplus S_{k+1-b}(t) & \text{if } 2 \leq b \leq k - 1. \end{cases} \end{aligned} \tag{5}$$

Note that in any of the equations that we have, we still have one term containing some sub-blocks of the ciphertext. To increase the probability that the equation is satisfied, we can set it to have no difference in any of the plaintext-ciphertext pairs. So in particular, in the CCA model, pick m different ciphertext such that:

If $b = 0$, pick all the ciphertext with fixed values of $S_0(s)$ and $S_{k-1}(s)$. Hence in the CCA attack, $m \leq 2^{(k-2)n}$.

If $b = 1, \dots, k-2$, fix the values of $S_{k-b}(s)$ and $S_{k-b-1}(s)$ for all $s = 0, \dots, m-1$. Again, in the CCA attack, $m \leq 2^{(k-2)n}$.

If $b = k - 1$ and $k \geq 4$, fix the values of $S_0(s), S_1(s)$ and $S_{k-1}(s)$ for $s = 0, \dots, m-1$. In this case, the CCA attack must have m to be at most $2^{(k-3)n}$.

So in summary, the differential trail for $r = ak + b$ rounds where $b \in \{0, \dots, k - 1\}$ is as follows:

1. In the KPA setting, the input (ciphertext) differential is $(\nabla_0, \dots, \nabla_{k-1})$ while the output (plaintext) differential is $(\Delta_0, \dots, \Delta_{k-1})$ where it satisfies the following equations:
 - $\Delta_0 = 0$.
 - $\Delta_1 = \nabla_{(k+1-b) \pmod k}$
 - All other sub-blocks difference is arbitrary, which we denote by \star .
2. In the CCA setting, the differential is the same, however, we impose some requirement to the ciphertext that we pick:
 - If $b = 0$, we fix the value of $S_0(s)$ and $S_{k-1}(s)$.
 - If $b = 1, \dots, k - 2$, the values of $S_{k-b}(s)$ and $S_{k-b-1}(s)$ are fixed.
 - If $b = k - 1$, we fix the values of $S_0(s), S_1(s)$ and $S_{k-1}(s)$.

Let $\mathcal{N}_{F,M}$ be the random variable representing the number of sets of two plaintext-ciphertext pairs that satisfy the conditions given by (5) for F representing the function used, which has value in the set $\{\text{perm}, \mathcal{F}\}$, and $M \in \{\text{KPA}, \text{CCA}\}$. $F = \text{perm}$ is used for the random permutation while $F = \mathcal{F}$ is used for the r -round Type-1 Feistel Scheme.

Now it is easy to see that the probability that the requirement set above to be true is equal to the probability that the right hand side of the equations to agree, which can be computed since we can assume all the X_i is uniformly and independently distributed by the ideality of the round function (which has been discussed in Sect. 3).

Calculating the expected values and variance of the random variables,

$$E(\mathcal{N}_{(\text{perm}, \text{KPA})}), E(\mathcal{N}_{(\text{perm}, \text{CCA})}), V(\mathcal{N}_{(\text{perm}, \text{KPA})}), V(\mathcal{N}_{(\text{perm}, \text{CCA})})$$

are all approximately $\frac{m^2}{2 \cdot 2^{2n}}$. Calculating the random variables corresponding to \mathcal{F} , the expected values and variances are summarised in Table 3 which can be found in Appendix A. The details on the calculation of the expected values and variances of $\mathcal{N}_{\mathcal{F}, \text{CCA}}$ for $b = 0$ can be found in the full version and is omitted here due to its similarity with the calculation done in [12]. The other results can be calculated using the same method.

Using the proposition of distinguishability of two random variables given in the preliminaries, the result is provided in Table 1.

In the KPA model, the maximum number of rounds is k^2 where from $k(k - 1) + 1$ up to k^2 rounds, the complexity is $\sqrt{2} \cdot 2^{(k-1)n}$. Furthermore, in the CCA model, the maximum number of rounds distinguishable is $k(k - 1) + k - 2 = k^2 - 2$ rounds with complexity $\sqrt{2} \cdot 2^{(k-3)n}$.

4.2 Comparison with Existing Result from [12]

To compare with the result given in [12] first note that there are some constant multipliers that are omitted in [12]. More specifically, all the expected values and variances should be multiplied by $\frac{1}{2}$. This constant adjustment comes from the

Table 1. Summary of distinguishability of Type-1 Feistel Scheme

b	Model	Complexity of distinguishing $ak + b$ rounds	Maximum a
0	KPA	$\sqrt{2} \cdot 2^{(a-1)n}$	k
	CCA	$\sqrt{2} \cdot 2^{(a-2)n}$	$k - 1$
$1 \leq b \leq k - 2$	KPA	$\sqrt{2} \cdot 2^{an}$	$k - 1$
	CCA	$\sqrt{2} \cdot 2^{(a-2)n}$	$k - 1$
$k - 1$	KPA	$\sqrt{2} \cdot 2^{an}$	$k - 1$
	CCA	$\sqrt{2} \cdot 2^{(a-2)n}$	$k - 2$

fact that given m plaintext-ciphertext pairs, the number of sets of 2 distinct pairs should be $\frac{m(m-1)}{2} \approx \frac{m^2}{2}$ instead of m^2 . Although the constant multiplier is very close to one compared to 2^n , it affects the maximum number of rounds that can be distinguished in the KPA and CPA model. This is because all the complexities of distinguishers should be multiplied by a factor of $\sqrt{2}$. The existence of this factor makes it impossible for a to reach the maximum number given in [12]. For $ak - 2$ rounds distinguished in KPA model, the complexity should be $\sqrt{2} \cdot 2^{(a-2)n}$. Hence the maximum number of rounds that can be distinguished in the KPA model is $k^2 + k - 2$ rounds instead of $k^2 + 2k - 2$ rounds. Similarly, for $ak - 1$ rounds to be distinguishable in CPA, the complexity is again $\sqrt{2} \cdot 2^{(a-2)n}$. Therefore, the maximum number of rounds that is distinguishable in CPA model to be $k^2 - 1$ rounds instead of $k^2 + k - 1$.

Note that in both cases, the maximum number of rounds distinguishable without any complexity constraint is still better in the forward direction. So in this section, the advantage of using the backward direction analysis in a constrained environment is discussed.

We compare the results in the CCA model presented above with the CPA model.

1. When the complexity is fixed to $\sqrt{2} \cdot 2^{tn}$, in CPA model, the maximum number of rounds that is distinguishable is $(t + 2)k - 1$ while in CCA model, the maximum number of rounds that is distinguishable is $(t + 2)k + (k - 2) = (t + 3)k - 2 = (t + 2)k - 1 + k - 1$ which is an increase of $k - 1$ rounds.
2. Suppose that we want to distinguish r rounds for some positive integer r . Table 3 of [12] (after the adjustment by a factor of $\sqrt{2}$) tells us that when $pk - (p - 2) = (p - 1)k + k - p + 2 \leq r \leq (p + 1)k - p = pk + k - p$, the complexity is $\sqrt{2} \cdot 2^{(p-2)n}$. Using the same bound for r , the complexity is $\sqrt{2} \cdot 2^{(p-3)n} = \sqrt{2} \cdot 2^{(p-2)n} \cdot 2^{-n}$ when $r \leq pk - 1$ and $\sqrt{2} \cdot 2^{(p-2)n}$ when $r \geq pk$ (see Table 1). So the complexity is reduced by a factor of $\frac{1}{2^n}$ when $(p - 1)k + k - p + 2 \leq r \leq pk$ for any value of p .

Now for all the following sections, since the method that is being used is exactly the same, we will not discuss in detail on how to choose the differential, the expected values and the distinguishability. Instead, only the final results will

be stated and compared. We note that since we are using Proposition 1, which is also used in the analysis in [12], has success probability at least 70%.

5 Type-3 Feistel Scheme

5.1 Analysis of the Type-3 Feistel Scheme

As before, we denote the input as S_0, \dots, S_{k-1} . Define intermediate variables X_i such that $(X_{tk}, \dots, X_{t(k-1)})$ is the state value after t rounds. Assuming the number of rounds is r , for $0 \leq s \leq k-1, X_s = S_s$ and $X_{r(k+s)} = I_s$. Given the input of round $c, 1 \leq c \leq r$, by definition:

$$X_{ck} = X_{(c-1)k+k-1}$$

$$X_{ck+s} = X_{(c-1)k+(s-1)} \oplus F_{(r+1-c, s-1)}(X_{ck+s-1}), \forall 1 \leq s \leq k-1.$$

Let $r = ak + b$ for $0 \leq b \leq k-1$. In this paper, we only consider $a = 1$ and $b > 0$. Expanding the equation for $X_{(k+b)k+s}$ using the equation given above, the following can then be derived:

- When $b = s$,

$$X_{(k+b)k+s} = \bigoplus_{i=0}^{b-1} F_{(i+1, s-1-i)}(X_{(k+b-i)k+(s-1-i)})$$

$$\oplus \bigoplus_{i=0}^{k-2} F_{(i+b+2, k-2-i)}(X_{(k-1-i)k+(k-2-i)}) \oplus S_0.$$

- When $b = s + 1 \leq k - 1$,

$$X_{(k+b)k+s} = \bigoplus_{i=0}^{s-1} F_{(i+1, s-1-i)}(X_{(k+b-i)k+(s-1-i)}) \oplus F_{(b+1, k-2)}(X_{(k)k+k-2})$$

$$\oplus \bigoplus_{i=0}^{k-3} F_{(i+b+2, k-3-i)}(X_{(k-1-i)k+k-3-i}) \oplus S_{k-1}.$$

- When $s + 1 < b \leq k - 1$,

$$X_{(k+b)k+s} = \bigoplus_{i=0}^{s-1} F_{(i+1, s-1-i)}(X_{(k+b-i)k+(s-1-i)})$$

$$\oplus \bigoplus_{i=0}^{b-s-1} F_{(s+i+2, k-2-i)}(X_{(k+b-s-1-i)k+k-2-i})$$

$$\oplus \bigoplus_{i=0}^{k-b+s-2} F_{(i+b+2, k-b+s-2-i)}(X_{(k-1-i)k+k-b+s-2-i})$$

$$\oplus \bigoplus_{i=0}^{b-s-2} F_{(k+s+i+2, k-2-i)}(X_{(b-s-1-i)k+(k-2-i)}) \oplus S_{k-1}.$$

- When $s = b + 1 \leq k - 1$,

$$X_{(k+t)k+b} = \bigoplus_{i=0}^b F_{(i+1,b-i)}(X_{(k+b-i)k+b-i}) \oplus \bigoplus_{i=0}^{k-3} F_{(i+3,k-2-i)} F(X_{(k-2-i)k+(k-2-i)} \oplus S_1).$$

- When $b + 1 < s \leq k - 1$,

$$X_{(k+b)k+s} = \bigoplus_{i=0}^b F_{(i+1,b-i)}(X_{(k+b-i)k+b-i}) \oplus \bigoplus_{i=0}^{s-b-2} F_{(b+i+2,s-b-2-i)}(X_{(k-1-i)k+s-b-2-i}) \oplus \bigoplus_{i=0}^{k-s+b-2} F_{(s+2+i,k-2-i)}(X_{(k-s+b-1-i)k+(k-2-i)}) \oplus S_{s-b}.$$

Let $b \in \{1, \dots, k-1\}$. For the m plaintext-ciphertext pairs, the distinguishing attack counts the number of sets of two pairs (j, j') , $1 \leq j < j' \leq m$ that satisfies the following two conditions:

1. $I_{(r-1)}(j) = I_{(r-1)}(j')$
 2. $I_r(j) \oplus I_r(j') = S_0(j) \oplus S_0(j')$.
- (6)

In the CCA model, fix the value of $S_{k-1}(j)$ of all the m ciphertexts. Hence $m \leq 2^{(k-1)n}$.

Calculating the random variables with the same method, $E(\mathcal{N}_{(\text{perm}, \text{KPA})})$, $V(\mathcal{N}_{(\text{perm}, \text{KPA})})$, $E(\mathcal{N}_{(\text{perm}, \text{CCA})})$, $V(\mathcal{N}_{(\text{perm}, \text{CCA})})$, $V(\mathcal{N}_{(\mathcal{F}, \text{KPA})})$ and $V(\mathcal{N}_{(\mathcal{F}, \text{CCA})})$ are all approximately $\frac{m^2}{2 \cdot 2^{2n}}$ while

$$E(\mathcal{N}_{(\mathcal{F}, \text{KPA})}) = \frac{m^2}{2} \left(\frac{1}{2^{2n}} + \frac{1}{2^{(k+r-2)n}} \right)$$

and

$$E(\mathcal{N}_{(\mathcal{F}, \text{CCA})}) = \frac{m^2}{2} \left(\frac{1}{2^{2n}} + \frac{1}{2^{(k+r-3)n}} \right).$$

In both KPA and CPA model, \mathcal{F} is distinguishable from a random permutation when there are up to $k+2$ rounds and the complexity to distinguish $k+b$ rounds are $\sqrt{2} \cdot 2^{(k+b-3)n}$ and $\sqrt{2} \cdot 2^{(k+b-4)n}$ respectively.

5.2 Comparison with Existing Result from [12]

Now we compare our result with the one given in [12]. First of all, note that in [12], there is a restriction that $\lfloor \frac{k}{2} \rfloor \geq 3$. This means that k needs to be at

least 6 while the distinguisher proposed above can be used for all $k \geq 2$. So in any attack model, this analysis provides a new lower bound of the maximum number of distinguishable round for $2 \leq k \leq 5$. Furthermore, when $k \geq 6$, in KPA model, in both $k + 1$ and $k + 2$ rounds proposed above, the complexity is higher than the one given in [12]. The same thing happen in the distinguisher for $k + 1$ rounds in the CCA model. However, the lower bound of maximum number of rounds distinguishable from random permutation in CCA model is increased to $k + 2$ from $k + 1$ proposed in [12].

6 Alternating Feistel Scheme

6.1 Analysis of Alternating Feistel Scheme

We divide this section into two cases based on the parity of the number of rounds. This is required due to the different round function in odd and even rounds.

Even Number of Rounds. Suppose that the number of rounds is $2r$. Let X_i be intermediate variables such that after $2t$ rounds the state value is

$$(X_{tk}, \dots, X_{tk+k-1}).$$

For any $0 \leq s \leq k-1$, $(I_s, S_s) = (X_{rk+s}, X_s)$. Then, given the state value after $2t$ rounds, $(X_{tk}, \dots, X_{tk+k-1})$ where $0 \leq t \leq r-1$, we have the following relations:

- $X_{(t+1)k} = X_{tk} \oplus F_{(r-t)}((X_{tk+s} \oplus F_{(r-t,s)}(X_{tk}))_{s=1}^{k-1})$ where

$$(Y_a)_{a=r}^s := (Y_r, Y_{r+1}, \dots, Y_s).$$

- $X_{(t+1)k+s} = X_{tk+s} \oplus F_{(r-t,s)}(X_{tk}), \forall s = 1, \dots, k-1.$

Then expand the equation for I_s :

- $I_0 = S_0 \oplus \bigoplus_{i=0}^{r-1} F_{(r-i)}((X_{ik+s} \oplus F_{(r-i,s)}(X_{ik}))_{s=1}^{k-1})$
- $\forall s \in \{1, \dots, k-1\},$

$$I_s = S_s \oplus \bigoplus_{i=0}^{r-1} F_{(r-i,s)}(X_{ik}) = S_s \oplus F_{(r,s)}(S_0) \oplus \bigoplus_{i=1}^{r-1} F_{(r-i,s)}(X_{ik}).$$

The distinguishing attack finds the number of sets of two plaintext-ciphertext pairs $(p, q), 1 \leq p < q \leq m$ such that they satisfy the following conditions:

$$\begin{aligned} 1. & \quad I_0(p) = I_0(q) \\ 2. & \quad \forall s = 1, \dots, k-1, I_b(p) \oplus I_b(q) = S_b(p) \oplus S_b(q). \end{aligned} \quad (7)$$

Furthermore, in the CCA model, all the ciphertexts are chosen such that they have the same fixed value in $S_0(p)$. So we have, $m \leq 2^{(k-1)n}$.

Using the same calculation as before, $E(\mathcal{N}_{(\text{perm}, \text{KPA})})$, $V(\mathcal{N}_{(\text{perm}, \text{KPA})})$, $E(\mathcal{N}_{(\text{perm}, \text{CCA})})$, $V(\mathcal{N}_{(\text{perm}, \text{CCA})})$, $V(\mathcal{N}_{(\mathcal{F}, \text{KPA})})$ and $V(\mathcal{N}_{(\mathcal{F}, \text{KPA})})$ can all be approximated by $\frac{m^2}{2 \cdot 2^{kn}}$. Furthermore,

$$E(\mathcal{N}_{(\mathcal{F}, \text{KPA})}) = \frac{m^2}{2} \left(\frac{1}{2^{kn}} + \frac{1}{2^{rn}} \right) \text{ and } E(\mathcal{N}_{(\mathcal{F}, \text{CCA})}) = \frac{m^2}{2} \left(\frac{1}{2^{kn}} + \frac{1}{2^{(r-1)n}} \right).$$

Simplifying this, to distinguish $2r$ rounds, the complexity is $\sqrt{2} \cdot 2^{(r-\frac{k}{2})n}$ for KPA and $\sqrt{2} \cdot 2^{(r-\frac{k}{2}-1)n}$ for CCA. So when k is even, in both models, \mathcal{F} can be distinguished from a random permutation when the round number is up to $3k-2$ with complexity $\sqrt{2} \cdot 2^{(k-1)n}$ and $\sqrt{2} \cdot 2^{(k-2)n}$ respectively. When k is odd, \mathcal{F} can be distinguished from a random permutation when the round number is up to $3k-1$. In this case, the complexity is $\sqrt{2} \cdot 2^{(k-\frac{1}{2})n}$ and $\sqrt{2} \cdot 2^{(k-\frac{3}{2})n}$ respectively.

Odd Number of Rounds. Suppose that the number of rounds is $2r+1$ for some non-negative integers r . Let X_i be intermediate variables such that for any non-negative integer t , after $2t+1$ rounds, the state value is $(X_{tk}, \dots, X_{tk+k-1})$. So $I_s = X_{rk+s}$ for all $0 \leq s \leq k-1$ while

$$X_s = \begin{cases} S_s, & \text{if } s = 1, \dots, k-1, \\ S_0 \oplus F_{r+1}(S_1, \dots, S_{k-1}) & \text{if } s = 0. \end{cases}$$

Following the expansion done before, the following equalities can be found:

- $I_0 = S_0 \oplus F_{r+1}(S_1, \dots, S_{k-1}) \oplus \bigoplus_{i=0}^{r-1} F_{(r-i)}((X_{ik+s} \oplus F_{(r-i,s)}(X_{ik}))_{s=1}^{k-1})$
- $\forall s = 1, \dots, k-1, I_s = S_s \oplus \bigoplus_{i=0}^{r-1} F_{(r-i,s)}(X_{ik}) = S_s \oplus F_{(r,s)}(S_0) \oplus \bigoplus_{i=1}^{r-1} F_{(r-i,s)}(X_{ik})$.

Because of this, all the distinguisher and calculation considered in the even number of rounds case can still be used in this case. Hence the complexity to distinguish $2r+1$ rounds is $\sqrt{2} \cdot 2^{(r-\frac{k}{2})n}$ for KPA and $\sqrt{2} \cdot 2^{(r-\frac{k}{2}-1)n}$ for CCA. When k is even, in both models, \mathcal{F} can be distinguished from a random permutation when the round number is up to $3k-1$ with complexity $\sqrt{2} \cdot 2^{(k-1)n}$ and $\sqrt{2} \cdot 2^{(k-2)n}$ respectively. When k is odd, we can distinguish up to $3k$ rounds with complexity $\sqrt{2} \cdot 2^{(k-\frac{1}{2})n}$ and $\sqrt{2} \cdot 2^{(k-\frac{3}{2})n}$ respectively.

6.2 Comparison with Existing Result from [12]

We compare the result of previous subsection with the one given in Sect. 4.4 in [12]. As before, note all the expected values and variance should be multiplied by $\frac{1}{2}$, all the complexities should be multiplied by $\sqrt{2}$ and hence the maximum number of rounds, in this case, should be decreased by 2. After this adjustment, to distinguish t rounds, the complexities are summarised in Table 2:

Table 2. Summary of comparison of Alternating Feistel Schemes with fixed number of rounds t .

Parity of t	Attack model	Complexity [12]	Complexity (This Paper)
Odd ($t = 2p - 1$)	KPA	$\sqrt{2} \cdot 2^{(p-\frac{k}{2})n} \cdot 2^{\frac{n}{2}}$	$\sqrt{2} \cdot 2^{(p-\frac{k}{2})n} \cdot 2^{-n}$
	CPA/CCA	$\sqrt{2} \cdot 2^{(p-\frac{k}{2})n} \cdot 2^{\frac{n}{2}}$	$\sqrt{2} \cdot 2^{(p-\frac{k}{2})n} \cdot 2^{-2n}$
Even ($t = 2p$)	KPA	$\sqrt{2} \cdot 2^{(p-\frac{k}{2})n}$	$\sqrt{2} \cdot 2^{(p-\frac{k}{2})n}$
	CPA/CCA	$\sqrt{2} \cdot 2^{(p-\frac{k}{2})n}$	$\sqrt{2} \cdot 2^{(p-\frac{k}{2})n} \cdot 2^{-n}$

In both models, when the number of rounds is odd, the complexity is better than the forward direction, which is a reduction by a factor of $2^{\frac{3n}{2}}$. However, when the number of rounds is even, backward direction requires the same complexity in the KPA model. In the CCA model, the complexity of backward direction is reduced by a factor of 2^n .

Note that after the adjustment to the result in [12], backward differential analysis achieves 2 more rounds in both models, from $3k - 2$ rounds to $3k$ rounds.

7 Unbalanced Feistel Scheme

7.1 Analysis of Unbalanced Feistel Scheme

In this section we only consider two special cases of $\text{UFN}(k', k)$. We discuss the analysis of the case when k is divisible by k' . This is a generalization of the UFN discussed in Sect. 6 of [16] where k is set to be 3 and k' is set to be 1. It can also be seen as a generalization of the UFN discussed in [17] where $k' = 1$. In Appendix D of the full version², the case when $k - k'$ is a factor of k is also considered. Due to the similarity of the technique used and also the page restriction, the detail of the analysis is omitted. This second case is a generalization of the analysis of $\text{UFN}(k', k)$ when $k' = k - 1$ in [8, 18, 23].

Analysis of $\text{UFN}(k', k)$ when k' divides k . Let A be a positive integer such that $k = Ak'$. Define intermediate variables X_i such that $(X_{sk}, \dots, X_{sk+(k-1)})$ is the state value after s rounds. So

$$(X_0, \dots, X_{k-1}) = (S_0, \dots, S_{k-1}).$$

Suppose that the number of rounds is $r = pA + q$ where $0 \leq q \leq A - 1$. So $(X_{rk}, \dots, X_{rk+k-1}) = (I_0, \dots, I_{k-1})$. Given the state value after $s - 1$ ($s \geq 1$) backward rounds $(X_{(s-1)k}, \dots, X_{(s-1)k+k-1})$, the output of the s -th backward round can be computed by:

- $X_{sk+t} = X_{(s-1)k+(t-k')}$ if $k' \leq t \leq k - 1$,

² The full version will be uploaded to Cryptology ePrint archive soon.

- $X_{sk+t} = X_{(s-1)k+(k-k'+t)} \oplus F_{r+1-s,t}(X_{(s-1)k}, \dots, X_{(s-1)k+k-k'-1}) = X_{(s-1)k+(k-k'+t)} \oplus F_{r+1-s,t}(X_{sk+k'}, \dots, X_{sk+k-1})$ if $0 \leq t \leq k' - 1$.
Now for $0 \leq t \leq k' - 1$, expanding the relation given above, we get:
 - If $q = 0$,

$$I_t = S_t \oplus \bigoplus_{s=1}^{p-1} F_{sA+2,t}(X_{(r-sA)k+k'}, \dots, X_{(r-sA)k+k-1}) \oplus F_{2,t}(I_{k'}, \dots, I_{k-1}),$$

- Otherwise,

$$I_t = S_{(A-q)k'+t} \oplus \bigoplus_{s=1}^p F_{sA+2,t}(X_{(r-sA)k+k'}, \dots, X_{(r-sA)k+k-1}) \oplus F_{2,t}(I_{k'}, \dots, I_{k-1}).$$

The distinguisher counts the number of set of two plaintext ciphertext pairs (i, j) , $1 \leq i < j \leq m$ such that

$$\forall t = 0, \dots, k' - 1, I_t(i) \oplus I_t(j) = S_{(a-q)k'+t}(i) \oplus S_{(a-q)k'+t}(j).$$

In the CPA model, pick ciphertexts with a fixed value in $I_{k'}, \dots, I_{k-1}$. In other words, the maximum number of plaintext-ciphertext pairs is $m \leq 2^{k'n}$.

The expected values and variances of the random variables can be found in Table 4 in Appendix A.

Using the definition of distinguishable, the complexity and maximum number of rounds distinguishable are summarised in Tables 5 and 6 which can be found in Appendix B.

A distinguisher for backward direction $\text{UFN}(k', k)$ can be constructed by considering the forward propagation of the equation. Hence, given the value of $X_{sk}, \dots, X_{sk+k-1}$, we have:

- $X_{sk+t} = X_{(s+1)k+(t+k')}$ if $0 \leq t \leq k - k' - 1$,
- $X_{sk+t} = X_{(s+1)k+(t-k+k')} \oplus F_{r-s,t}(X_{(s+1)k+k'}, \dots, X_{(s+1)k+k-1}) = X_{(s+1)k+(t-k+k')} \oplus F_{r-s,t}(X_{sk}, \dots, X_{sk+k-k'-1})$ if $k - k' \leq t \leq k - 1$.

Expanding S_t , the following equalities can be obtained:

- If $q = 0$,

$$S_t = I_t \oplus \bigoplus_{s=1}^{p-1} F_{r-sA,t}(X_{sAk}, \dots, X_{sAk+k-k'-1}) \oplus F_{r,t}(S_0, \dots, I_{k-k'-1}),$$

- Otherwise,

$$S_t = I_{t-(A-q)k'} \oplus \bigoplus_{s=1}^p F_{r-sA,t}(X_{iAk}, \dots, X_{iAk+k-k'-1}) \oplus F_{r,t}(S_0, \dots, S_{k-k'-1}).$$

The distinguisher finds the number of sets of two plaintext-ciphertext pairs (i, j) such that $\forall t = k - k', \dots, k - 1, S_t(i) \oplus S_t(j) = I_{t-(A-q)k'}(i) \oplus I_{t-(A-q)k'}(j)$ where $S_0, \dots, S_{k-k'-1}$ are fixed in the CCA model. It is easy to see that with this model, we have exactly the same expected values, variances and distinguishability as the ones found in Tables 4, 5 and 6.

8 Conclusion

In this paper, differential analysis on the inverse function of four different types of generic Generalized Feistel Scheme, namely Type-1, Type-3, Alternating Scheme and $\text{UFN}(k', k)$ was considered. We show that for Type-1 Feistel Scheme, backward distinguisher performs better especially in the chosen ciphertext attack compared to the results in [12]. Using the same complexity, we can distinguish $k - 1$ more rounds while distinguishing the same number of rounds requires smaller complexity with factor of $\frac{1}{2^n}$.

In Type-2 and Alternating Feistel scheme, although there are some difference in the complexity, both directions can achieve almost the same number of rounds. This shows that these two types can be seen as almost symmetric from both direction.

We improve the differential cryptanalysis in Type-3 Feistel Scheme in several cases. In the KPA model with low number of branches, $2 \leq k \leq 5$, our analysis provides a lower bound of the number of rounds that is indistinguishable from random permutation. Secondly, in the CCA model, the lower bound of maximum number of rounds distinguishable is increased by 1 round, from $k + 1$ obtained in [12] to $k + 2$.

In Alternating Feistel Scheme, we achieve 2 more rounds than the one claimed in [12]. The complexity is reduced by a factor of $2^{\frac{3n}{2}}$ when distinguishing the same odd number of rounds.

Lastly, a lower bound for the maximum number of rounds that is distinguishable from random permutation in $\text{UFN}(k', k)$ scheme is given through the forward direction distinguisher. To the best of our knowledge, this is the first bound given in a rather general case in which k' is arbitrary as long as k' is a divisor of k for any integer k .

A Expected Value and Variance of Random Variables Concerning Type-1 Feistel Scheme and $\text{UFN}(k', k)$ When k' Divides k .

The following table summarises the expected value and variance of the random variables used in the analysis of Type-1 Feistel Schemes.

The next table summarises the expected values and variances for random variables used in the analysis of $\text{UFN}(k', k)$ when k' divides k .

Table 3. Expected value and variance of random variables concerning Type-1 Feistel Schemes

b	Attack model	Expected value	Variance	Maximum value of m
0	KPA	$\frac{m^2}{2} \left(\frac{1}{2^{2n}} + \frac{1}{2^{an}} \right)$	$\frac{m^2}{2 \cdot 2^{2n}}$	2^{kn}
	CCA	$\frac{m^2}{2} \left(\frac{1}{2^{2n}} + \frac{1}{2^{(a-1)n}} \right)$	$\frac{m^2}{2 \cdot 2^{2n}}$	$2^{(k-2)n}$
$1 \leq b \leq k-2$	KPA	$\frac{m^2}{2} \left(\frac{1}{2^{2n}} + \frac{1}{2^{(a+1)n}} \right)$	$\frac{m^2}{2 \cdot 2^{2n}}$	2^{kn}
	CCA	$\frac{m^2}{2} \left(\frac{1}{2^{2n}} + \frac{1}{2^{(a-1)n}} \right)$	$\frac{m^2}{2 \cdot 2^{2n}}$	$2^{(k-2)n}$
$k-1$	KPA	$\frac{m^2}{2} \left(\frac{1}{2^{2n}} + \frac{1}{2^{(a+1)n}} \right)$	$\frac{m^2}{2 \cdot 2^{2n}}$	2^{kn}
	CCA	$\frac{m^2}{2} \left(\frac{1}{2^{2n}} + \frac{1}{2^{(a-1)n}} \right)$	$\frac{m^2}{2 \cdot 2^{2n}}$	$2^{(k-3)n}$

Table 4. Expected value and variance for various cases of UFN(k', k)

Attack model	q value	Π	E	V	σ
KPA	0	Perm	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m}{\sqrt{2} \cdot 2^{\frac{k'}{2}n}}$
		\mathcal{F}	$\frac{m^2}{2} \cdot \left(\frac{1}{2^{k'n}} + \frac{1}{2^{(k'+p-1)n}} \right)$	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m}{\sqrt{2} \cdot 2^{\frac{k'}{2}n}}$
	$1 \leq q \leq A-1$	Perm	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m}{\sqrt{2} \cdot 2^{\frac{k'}{2}n}}$
		\mathcal{F}	$\frac{m^2}{2} \cdot \left(\frac{1}{2^{k'n}} + \frac{1}{2^{(k'+p)n}} \right)$	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m}{\sqrt{2} \cdot 2^{\frac{k'}{2}n}}$
CPA	0	Perm	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m}{\sqrt{2} \cdot 2^{\frac{k'}{2}n}}$
		\mathcal{F}	$\frac{m^2}{2} \cdot \left(\frac{1}{2^{k'n}} + \frac{1}{2^{(k'+p-2)n}} \right)$	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m}{\sqrt{2} \cdot 2^{\frac{k'}{2}n}}$
	$1 \leq q \leq A-1$	Perm	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m}{\sqrt{2} \cdot 2^{\frac{k'}{2}n}}$
		\mathcal{F}	$\frac{m^2}{2} \cdot \left(\frac{1}{2^{k'n}} + \frac{1}{2^{(k'+p-1)n}} \right)$	$\frac{m^2}{2 \cdot 2^{k'n}}$	$\frac{m}{\sqrt{2} \cdot 2^{\frac{k'}{2}n}}$

B Distinguishability Table for UFN(k', k)

The following tables contain the summary of distinguishability of UFN(k', k) from a random permutation.

Table 5. Complexity of Unbalanced Feistel Scheme

k'	q value	Attack model	Complexity of distinguishing $pA + q$ rounds
1	0	KPA	$\sqrt{2} \cdot 2^{(p-\frac{1}{2})n}$
		CPA/CCA	$\sqrt{2} \cdot 2^{(p-\frac{3}{2})n}$
	$1 \leq q \leq A-1$	KPA	$\sqrt{2} \cdot 2^{(p+\frac{1}{2})n}$
		CPA/CCA	$\sqrt{2} \cdot 2^{(p-\frac{1}{2})n}$
$k' \geq 2$	0	KPA	$\frac{\sqrt{2}}{k'} \cdot 2^{(\frac{k'}{2}+p-1)n}$
		CPA/CCA	$\frac{\sqrt{2}}{k'} \cdot 2^{(\frac{k'}{2}+p-2)n}$
	$1 \leq q \leq A-1$	KPA	$\frac{\sqrt{2}}{k'} \cdot 2^{(p+\frac{k'}{2})n}$
		CPA/CCA	$\frac{\sqrt{2}}{k'} \cdot 2^{(p+\frac{k'}{2}-1)n}$

Table 6. Summary of distinguishability of Unbalanced Feistel Scheme

k'	q value	Attack model	Maximum p distinguishable \rightarrow Maximum round distinguishable
1	0	KPA	$k \rightarrow k^2$
		CPA/CCA	$2 \rightarrow 2k$
	$1 \leq q \leq A - 1$	KPA	$k \rightarrow k^2 + k - 1$
		CPA/CCA	$1 \rightarrow k + k - 1$
$k' \geq 2$	0	KPA	$\begin{cases} k - \frac{k'}{2} + 1 \rightarrow \frac{k^2}{k'} - \frac{k}{2} + \frac{k}{k'} & \text{if } k' \text{ is even} \\ k - \frac{k'-1}{2} \rightarrow \frac{k^2}{k'} - \frac{k(k'-1)}{2k'} & \text{if } k' \text{ is odd} \end{cases}$
		CPA/CCA	$\begin{cases} \frac{k'}{2} + 2 \rightarrow \frac{k}{2} + 2\frac{k}{k'} & \text{if } k' \text{ is even} \\ \frac{k'+3}{2} \rightarrow \frac{k(k'+3)}{2k'} & \text{if } k' \text{ is odd} \end{cases}$
	$1 \leq q \leq A - 1$	KPA	$\begin{cases} k - \frac{k'}{2} \rightarrow \frac{k^2}{k'} - \frac{k}{2} + \frac{k}{k'} - 1 & \text{if } k' \text{ is even} \\ k - \frac{k'+1}{2} \rightarrow \frac{k^2}{k'} - \frac{k(k'+1)}{2k'} + \frac{k}{k'} - 1 & \text{if } k' \text{ is odd} \end{cases}$
		CPA/CCA	$\begin{cases} \frac{k'}{2} + 1 \rightarrow \frac{k}{2} + \frac{k}{k'} + \frac{k}{k'} - 1 & \text{if } k' \text{ is even} \\ \frac{k'+1}{2} \rightarrow \frac{k(k'+1)}{2k'} + \frac{k}{k'} - 1 & \text{if } k' \text{ is odd} \end{cases}$

References

- Anderson, R., Biham, E.: Two practical and provably secure block ciphers: BEAR and LION. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 113–120. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60865-6_48
- Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: *Camellia*: a 128-bit block cipher suitable for multiple platforms — design and analysis. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 39–56. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44983-3_4
- Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak Sponge Function Family Main Document. Submission to NIST (Round 2) (2009)
- 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
- 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
- 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
- Hoang, V.T., Rogaway, P.: On generalized feistel networks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 613–630. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_33
- Jutla, C.S.: Generalized birthday attacks on unbalanced Feistel networks. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 186–199. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0055728>
- Knudsen, L.: DEAL - a 128-bit block cipher. In: NIST AES Proposal (1998)
- Luby, M., Rackoff, C.: How to construct pseudo-random permutations from pseudo-random functions. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 447–447. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_34

11. Lucks, S.: Faster Luby-Rackoff ciphers. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 189–203. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60865-6_53
12. Nachev, V., Volte, E., Patarin, J.: Differential attacks on generalized Feistel schemes. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. 8257, pp. 1–19. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02937-5_1
13. Nyberg, K.: Generalized Feistel networks. In: Kim, K., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 91–104. Springer, Heidelberg (1996). <https://doi.org/10.1007/BFb0034838>
14. Patarin, J.: Generic attacks on Feistel schemes. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 222–238. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_14
15. Patarin, J.: Security of random feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 106–122. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_7
16. Patarin, J.: Security of Balanced and Unbalanced Feistel Schemes with Linear Non Equalities. Cryptology ePrint Archive, Report 2010/293 (2010). <http://eprint.iacr.org/2010/293>
17. Patarin, J., Nachev, V., Berbain, C.: Generic attacks on unbalanced feistel schemes with contracting functions. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 396–411. Springer, Heidelberg (2006). https://doi.org/10.1007/11935230_26
18. Patarin, J., Nachev, V., Berbain, C.: Generic attacks on unbalanced Feistel schemes with expanding functions. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 325–341. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76900-2_20
19. Schneier, B., Kelsey, J.: Unbalanced Feistel networks and block cipher design. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 121–144. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60865-6_49
20. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit block-cipher CLEFIA (extended abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74619-5_12
21. Tjuawinata, I., Huang, T., Wu, H.: Cryptanalysis of simpira v2. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 2017. LNCS, vol. 10342, pp. 384–401. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60055-0_20
22. Treger, J., Patarin, J.: Generic attacks on feistel networks with internal permutations. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 41–59. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02384-2_4
23. Volte, E., Nachev, V., Patarin, J.: Improved generic attacks on unbalanced feistel schemes with expanding functions. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 94–111. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_6
24. Zheng, Y., Matsumoto, T., Imai, H.: On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 461–480. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_42

Improvements for Gate-Hiding Garbled Circuits

Mike Rosulek^(✉)

Oregon State University, Corvallis, USA
rosulekm@eecs.oregonstate.edu

Abstract. Garbled circuits have been highly optimized for practice over the last several years. Today’s most efficient constructions treat different types of gates (e.g., AND vs. XOR) differently; as such, they leak the type of each gate. In many applications of garbled circuits, the circuit itself is public, so such leakage is tolerable. In other settings, however, it is desirable to hide the type of each gate.

In this paper we consider optimizing garbled circuits for the gate-hiding case. We observe that the best state-of-the-art constructions support only a limited class of gate functions, which turns out to undermine their improvements in several settings. These state-of-the-art constructions also require a non-minimal hardness assumption.

We introduce two new gate-hiding constructions of garbled circuits. Both constructions achieve the same communication complexity as the best state-of-the-art schemes, but support a more useful class of boolean gates and use only the minimal assumption of a secure PRF.

1 Introduction

Garbled circuits were first proposed by Yao in the 1980s [25] and have since become the target of many improvements. Garbled circuits form the basis of secure two-party computation protocols and many other applications in cryptography.

In a typical scenario involving two-party computation, both parties agree on some circuit f that they wish to evaluate. Since f is public, the garbled circuits in these protocols do not need to hide anything about f ; they need to hide only the *inputs* to f . However, in some applications like private function evaluation (PFE) [1, 10, 18] it is useful for the garbled circuit to hide information about the circuit itself.

In this work, we study garbled circuit constructions that are **gate-hiding**—that is, they leak only the topology of the circuit, while hiding the type of each gate (e.g., AND, XOR, NOR).

Comparing efficiency of garbling schemes. With the ubiquity of native AES instructions on modern CPUs, applications of garbled circuits are rarely CPU-bound but are usually network-bound (cf. [26]). Hence, the most important metric

Partially supported by NSF awards 1149647 & 1617197.

for measuring the efficiency of garbled circuits is their **size** (typically bits per gate). In all of the schemes we will discuss, the difference in garbled circuit size is only in the constant factors. All these schemes produce garbled gates with size $c\lambda + d$, where c and d are small constants and λ is the computational security parameter.

1.1 State of the Art

It is folklore that the “textbook” Yao garbling scheme is gate-hiding. Indeed, in the security proof for Yao’s protocol from [16], there is a hybrid in which each garbled gate is replaced by a garbled “always-zero” gate. Since every garbled gate is indistinguishable from such a constant gate, the scheme hides the type of gate. The same property holds for simple improvements like *garbled row reduction* (GRR3) [19]; that is, they hide the type of each gate. These two schemes garble each gate at a cost of $4\lambda + 4$ and $3\lambda + 4$ bits, respectively.¹

The most efficient constructions of garbled circuits are derived from the Free XOR optimization [15] (including [7, 14, 26]). These constructions are **not** gate-hiding because the evaluator must behave very differently for XOR gates vs. non-XOR gates. This is what allows XOR gates to be garbled more efficiently than other gates in these constructions. These schemes therefore leak whether a gate is XOR or not, while typically hiding all further distinctions.

Two recent papers, one by Kempka et al. [11] (hereafter KKS) and one by Wang and Malluhi [24] (hereafter WM), each describe a gate-hiding garbling scheme in which each garbled gate costs only $2\lambda + O(1)$ bits. Both schemes use the same representation of truth values as garbled wire labels, but otherwise use very different techniques for garbled gates.² These two schemes are currently the most lightweight gate-hiding schemes.³ We discuss them in more detail in Sect. 3.

A summary of the state of the art for gate-hiding garbling schemes is given in Fig. 1.

1.2 What Kinds of Gates Are Supported, and Why Does It Matter?

Closer inspection of Fig. 1 reveals that these constructions are not entirely interchangeable. In particular, *they support different classes of gate functionalities*. We identify three different classes of gate functionalities below:

¹ The “extra” 4 bits come from using the point-permute optimization. In practice one would typically use the underlying cryptographic primitive (e.g., AES) in a way that gives security $\lambda = 127$, and then the garbled gates become a clean multiple of 128 bits in length. All of the constructions in this work use the point-permute optimization, which we discuss in greater detail in Sect. 2.4.

² KKS also show how to reduce the size of garbled gates at the input layer of a circuit. In this work we focus on internal gates of a circuit, and assume that the input gates represent only a small fraction of the circuit.

³ In this paper we restrict our attention to constructions based on symmetric-key primitives only. There exist garbled circuit constructions based on very expensive primitives (functional encryption, FHE) where the cost of every garbled gate is constant.

	size (bits)	gate basis	garble cost		eval. cost		assump.
			H	interp	H	interp	
Textbook Yao	$4\lambda + 4$	\mathcal{G}_{all}	4	0	1	0	PRF
GRR3 [19]	$3\lambda + 4$	\mathcal{G}_{all}	4	0	1	0	PRF
KKS [11]	$2\lambda + 8$	$\mathcal{G}_{\text{symm}}$	3	0	1	0	circ+RK
WM [24]	$2\lambda + 2$	$\mathcal{G}_{\text{symm}}$	3	1	1	1	circ+RK
KKS ^{+unary} (§3)	$3\lambda + 8$	\mathcal{G}_{all}	5	0	2	0	circ+RK
WM ^{+unary} (§3)	$3\lambda + 2$	\mathcal{G}_{all}	5	1	2	1	circ+RK
new (§4)	$2\lambda + 4$	$\mathcal{G}_{\text{non-const}}$	4	2	1	1	PRF
new (§5)	$2\lambda + 8$	$\mathcal{G}_{\text{non-const}}$	4	0	1	0	PRF

Fig. 1. Comparison of gate-hiding garbling schemes. All costs are listed per-gate. “ H ” refers to calls to a symmetric-key primitive (see Sect. 2.3 for the required primitive). “interp” refers to interpolations of degree-2 polynomials over $GF(2^\lambda)$. “circ+RK” refers to a circularity+related-key assumption on H .

- \mathcal{G}_{all} : all gates $g : \{0, 1\}^2 \rightarrow \{0, 1\}$
- $\mathcal{G}_{\text{symm}}$: all gates satisfying $g(0, 1) = g(1, 0)$
- $\mathcal{G}_{\text{non-const}}$: all gates except for the degenerate ones $(a, b) \mapsto 0$ and $(a, b) \mapsto 1$

As mentioned above, it is not hard to see that textbook Yao and GRR3 support \mathcal{G}_{all} -gates and are gate-hiding with respect to this class. That is, they can garble literally any boolean gate functionality, in a way that hides the choice of gate.

However, the KKS and WM schemes—the schemes with smallest garbled gate size—**support only $\mathcal{G}_{\text{symm}}$ -gates**. While this seems like a minor limitation, we point out two reasons it can be problematic:

NOT gates. When dealing with fan-in-2 gates, one can usually think of any unary NOT gates being “absorbed” into all downstream (in the direction of evaluation) gates.⁴ This leads to non-symmetric gates like $(a, b) \mapsto a \wedge \bar{b}$. Almost all garbling schemes support “absorbed” negations at no additional cost, and in a way that hides the presence of the negations. Unfortunately, the glaring exceptions to this rule are in fact the KKS and WM schemes, which cannot garble these non-symmetric gates at all (we elaborate in Sect. 3)!

This raises the question of how to deal with a circuit containing NOT gates. While it is possible to express any circuit just in terms of NAND gates (which are symmetric), one will obtain smaller circuits by using a larger class of gates. Indeed, most of the available boolean circuits used for MPC are expressed as AND/XOR/NOT gates [9, 22]. In Sect. 3 we argue that NOT gates inherently have extra cost in the KKS & WM schemes, due to the structure of wire labels in these schemes. In contrast, a garbling scheme that is gate-hiding for $\mathcal{G}_{\text{non-const}}$ would not suffer from the same limitation, since this class of gates is closed under “absorbed” NOT gates.

⁴ Absorbing the NOT gate into its “upstream” gate may not always work, since the upstream gate may have multiple fan-out.

In short, closure under “absorbed NOT gates” is important since it implies that NOT gates will always be free. But schemes that support only $\mathcal{G}_{\text{symm}}$ do not have this property.

Multiplexers, pass-through gates. A multiplexer has three inputs and computes $(x_0, x_1, s) \mapsto x_s$. A pass-through gate is simply a multiplexer whose selection input s has been fixed, corresponding to either $(a, b) \mapsto a$ or $(a, b) \mapsto b$. These kinds of gates are clearly not in $\mathcal{G}_{\text{symm}}$, but are vital in most applications of gate-hiding garbled circuits. Garbling a pass-through gate in a gate-hiding way is therefore equivalent to garbling a multiplexer with its selection bit secretly fixed by the garbler. This is an approach used by Paus et al. [20] in an application to semi-private function evaluation, where the function being evaluated is hidden within a known class of functions. Constructions of *universal circuits* [8, 13, 17, 23] likewise use significant amount of multiplexers, and when the garbler is the party who programs the universal circuit, the multiplexer is meant to be replaced by a pass-through gate. Other variants of universal circuits [12] explicitly use pass-through gates as a fundamental concept in their constructions.

1.3 Hardness Assumptions

In the non-gate-hiding setting, the best garbling schemes use the Free XOR optimization [15] to eliminate communication for XOR gates. Choi et al. [6] showed that the Free XOR construction inherently relies on a nonstandard assumption: namely, that the underlying symmetric-key primitive have circular security and related-key security. For comparison, the minimal hardness assumption for garbled circuits is the existence of a PRF (equivalently, the existence of a one-way function). The best known way for garbling XOR gates from standard assumptions is due to Gueron et al. [7], who garble XOR gates at a cost of λ bits each, using a standard PRF.

It is reasonable to use a stronger assumption to achieve efficiency that we do not know how to achieve otherwise. In the case of non-gate-hiding garbled circuits, a nonstandard assumption allows XOR gates to be garbled for free.

However, in the gate-hiding case we cannot expect to garble XOR gates for free, since we cannot expect to garble *all* gates for free (for evidence, see the lower bound of Zahur et al. [26]). When XOR gates are not free, it is not clear that a stronger hardness assumption is necessary. Yet the best existing gate-hiding schemes (KKS and WM) both rely on a free-XOR-like hardness assumption that involves correlated keys and circularity. A natural question is whether this flavor of assumption is necessary for highly efficient, gate-hiding garbled circuits.

1.4 Our Contributions

In this work we present three main results:

In Sect. 3 we show how to extend the KKS & WM constructions from a $\mathcal{G}_{\text{symm}}$ -scheme to a \mathcal{G}_{all} -scheme. This comes at a price, however: the garbled gates must

increase in size from $2\lambda + O(1)$ to $3\lambda + O(1)$, and evaluation requires an extra cryptographic operation. We give evidence that this extra cost is inevitable—i.e., the KKS & WM approaches cannot be extended beyond symmetric gates for free.

In Sect. 4 we revisit a scheme of Pinkas et al. [21] that garbles non-constant gates ($\mathcal{G}_{\text{non-const}}$) for $2\lambda + 4$ bits. Their scheme, however, uses different methods to garble gates of even vs. odd parity (a gate has even parity if the number of 1's in its truth table is even). That is, the construction leaks the parity of a gate. We show that their odd-parity method can be adapted to work for even-parity gates as well, resulting in a gate-hiding scheme for $\mathcal{G}_{\text{non-const}}$ -gates with cost $2\lambda + 4$ bits per gate. While evaluation requires only a single cryptographic operation, it requires an additional polynomial interpolation step over $GF(2^\lambda)$, which in practice can cost roughly half of an AES evaluation (cf. [7]).

In Sect. 5 we present a new and novel gate-hiding scheme, inspired by a garbling technique of Gueron et al. [7]. This scheme supports $\mathcal{G}_{\text{non-const}}$ -gates, and results in garbled gates of size $2\lambda + 8$ bits. Evaluation involves just one cryptographic operation, and otherwise uses only XOR operations (in particular, no interpolation or finite field multiplications).

Our new constructions improve the concrete cost of applications of gate-hiding garbled circuits, by supporting circuits expressed over a more natural class of gates. Additionally, our new constructions require only the minimal hardness assumption of the existence of a PRF. As mentioned above, KKS & WM require a circularity/related-key assumption.

2 Preliminaries

2.1 Circuits

We represent circuits in the following way. All wires in the circuit (including input wires) are indexed in a topological ordering. For a circuit f , we define:

- $\text{inputs}(f)$: the set of indices of input wires
- $\text{gates}(f)$: the set of indices of non-input wires (i.e., wires that emanate from some internal gate)
- $\text{outputs}(f)$: the set of indices of output wires (not necessarily disjoint from the other sets).

For each gate with index $i \in \text{gates}(f)$, we define:

- $\text{left}(i)$: the index of the gate's left input wire
- $\text{right}(i)$: the index of the gate's right input wire
- $\text{type}(i)$: the functionality of the gate; i.e., a function $g : \{0, 1\}^2 \rightarrow \{0, 1\}$

For a circuit f , we let $\text{topo}(f)$ denote all of the above information except for $\text{type}(i)$.

Let \mathcal{G} be a set of boolean gates (e.g., \mathcal{G}_{all} , $\mathcal{G}_{\text{symm}}$). We say that a circuit f is a \mathcal{G} -circuit if $\text{type}(i) \in \mathcal{G}$ for every $i \in \text{gates}(f)$.

2.2 Garbled Circuits

We use the security definitions of Bellare et al. [4]. A **garbling scheme** is a collection of algorithms (Garble , Encode , Eval , Decode), with the following semantics:

- $\text{Garble}(1^\lambda, f) \rightarrow (F, e, d)$, where f is a boolean circuit, F is a garbled circuit, e is input-encoding information, and d is output-decoding information. Garble is a randomized algorithm, but the others are deterministic.
- $\text{Encode}(e, x) \rightarrow X$, where x is a plaintext circuit input, and X is a corresponding garbled input.
- $\text{Eval}(F, X) \rightarrow Y$, where Y is a garbled output.
- $\text{Decode}(d, Y) \rightarrow y$, where y is a plaintext output.

We say that the garbling scheme is a \mathcal{G} -**scheme**, if it supports f that are \mathcal{G} -circuits.

Several properties are useful, and here we state the relevant security properties for the case of **gate-hiding** \mathcal{G} -schemes:

- **Correctness:** For all $(F, e, d) \leftarrow \text{Garble}(1^\lambda, f)$, we have

$$\text{Decode}(d, \text{Eval}(F, \text{Encode}(e, x))) = f(x).$$

- **Gate-Hiding Privacy:** There exists a simulator \mathcal{S} , such that for all \mathcal{G} -circuits f and all inputs x , the following two distributions are indistinguishable:

$\begin{array}{l} \text{PrivReal}(1^\lambda, f, x): \\ \hline (F, e, d) \leftarrow \text{Garble}(1^\lambda, f) \\ X := \text{Encode}(e, x) \\ \text{return } (F, X, d) \end{array}$	$\begin{array}{l} \text{PrivSim}^{\mathcal{S}}(1^\lambda, f, x): \\ \hline \text{return } \mathcal{S}(1^\lambda, \text{topo}(f), f(x)) \end{array}$
---	---

In other words, (F, X, d) leaks no information beyond $\text{topo}(f)$ and $f(x)$.

- **Gate-Hiding Obliviousness:** There exists a simulator \mathcal{S} , such that for all \mathcal{G} -circuits f and all inputs x , the following two distributions are indistinguishable:

$\begin{array}{l} \text{OblivReal}(1^\lambda, f, x): \\ \hline (F, e, d) \leftarrow \text{Garble}(1^\lambda, f) \\ X := \text{Encode}(e, x) \\ \text{return } (F, X) \end{array}$	$\begin{array}{l} \text{OblivSim}^{\mathcal{S}}(1^\lambda, f, x): \\ \hline \text{return } \mathcal{S}(1^\lambda, \text{topo}(f)) \end{array}$
---	--

In other words, (F, X) (without d) leaks no information beyond $\text{topo}(f)$.

- **Authenticity:** For any \mathcal{G} -circuit f , input x , and efficient adversary \mathcal{A} , the following game outputs 1 with negligible probability:

$\begin{array}{l} \text{Auth}^{\mathcal{A}}(1^\lambda, f, x): \\ \hline (F, e, d) \leftarrow \text{Garble}(1^\lambda, f) \\ X := \text{Encode}(e, x) \\ \tilde{Y} \leftarrow \mathcal{A}(F, X) \\ \text{if } \text{Decode}(d, \tilde{Y}) \notin \{f(x), \perp\} \text{ then return 1 else return 0} \end{array}$
--

2.3 Dual-Key Hash

Our constructions require a function with type $H : \{0, 1\}^* \times \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\ell$. To define the required notion of security, we introduce a related function:

$$\mathcal{O}_K(t, a, X) = \begin{cases} H(t; X, K) & \text{if } a = 0 \\ H(t; K, X) & \text{if } a = 1 \end{cases}$$

We say that H is a **dual-key hash** if, for random choice of $K \leftarrow \{0, 1\}^\lambda$, oracle access to \mathcal{O}_K is indistinguishable from oracle access to a random function, against distinguishers who query \mathcal{O}_K with distinct t -values.

Intuitively, one can think of H as a PRF with two keys. The outputs of H look random as long as one of the two keys is random and secret (the other key can be chosen by the adversary). This notion is similar to the “dual PRF” definition in [5], however the additional and non-repeating t -input in our setting makes realizing our notion easier.

One can instantiate H as a random oracle. In practice, one might simply use $H = \text{sha256}$. In the standard model, one can use $H(t; A, B) = F(A, t) \oplus F(B, t)$, where F is a secure PRF. The fastest garbling schemes in practice use *fixed-key AES* as an ideal permutation, following [3], to take best advantage of AES hardware support. We think it likely that the schemes in this work can be adapted naturally to this setting. But since our focus is in part to minimize the hardness assumption of the schemes, we focus on the dual-key hash abstraction which can be instantiated from a plain PRF.

2.4 Basics of Garbled Circuit Techniques

We review several basic and standard techniques for garbled circuits. Readers familiar with the internals of recent garbled circuit constructions can skip this section.

Textbook Yao. Suppose a gate has input wire labels (A_0, A_1) and (B_0, B_1) , and output wire labels (C_0, C_1) . Here the subscripts correspond to the *truth value* (so A_0 is the wire label encoding false on that wire). Suppose we wish to garble an AND gate as an example, then we generate the following encryptions:

$$\begin{array}{ll} G_1 = \text{Enc}(A_0, \text{Enc}(B_0, C_0)) & G_3 = \text{Enc}(A_1, \text{Enc}(B_0, C_0)) \\ G_2 = \text{Enc}(A_0, \text{Enc}(B_1, C_0)) & G_4 = \text{Enc}(A_1, \text{Enc}(B_1, C_1)) \end{array}$$

However, position in this list clearly leaks the truth value on each wire. So the list is randomly permuted. The evaluator, who receives just a single garbled input combination A_a, B_b is expected to perform trial decryption of each ciphertext. Hence, the encryption scheme must give some indication of whether decryption is successful.

Point-permute. Beaver et al. [2] introduced a point-and-permute technique that is now used in essentially every practical garbling scheme. The idea is to append to each wire label a **color bit**. The two wire labels on each wire will have opposite color bits. The association between color bits and truth values is random and known only to the garbler. The evaluator, who sees only one label per wire, sees only a random color bit that is distributed independently of the truth value it represents.

As before, let a gate have input wire labels (A_0, A_1) and (B_0, B_1) , and output wire labels (C_0, C_1) . Unlike before, let the subscript denote the *color bit* of the wire label. The evaluator holds one label from each wire, and is allowed to use their (public) color bits to decide how to proceed. Suppose for example that A_0 , B_1 , and C_1 correspond to true on their respective wires. Then the garbled gate consists of four ciphertexts:

$$\begin{aligned} G_1 &= \text{Enc}(A_0, \text{Enc}(B_0, C_0)) & G_3 &= \text{Enc}(A_1, \text{Enc}(B_0, C_0)) \\ G_2 &= \text{Enc}(A_0, \text{Enc}(B_1, C_1)) & G_4 &= \text{Enc}(A_1, \text{Enc}(B_1, C_0)) \end{aligned}$$

The ciphertexts can be arranged in precisely this order, since the order depends only on the (public) color bits and not the (secret) truth values. The evaluator can use the color bits to identify exactly which ciphertext to decrypt. There is no need for the evaluator to perform trial decryption on each ciphertext, and therefore no need to detect “correct decryption.” This allows the scheme to use a simple encryption, namely:

$$\begin{aligned} G_1 &= H(A_0, B_0) \oplus C_0 & G_3 &= H(A_1, B_0) \oplus C_0 \\ G_2 &= H(A_0, B_1) \oplus C_1 & G_4 &= H(A_1, B_1) \oplus C_0 \end{aligned}$$

Here H is a dual-key hash, defined in Sect. 2.3 (a unique nonce should also be given as input to each invocation of H , but we have omitted it from the notation).

Simple garbled row reduction. Naor et al. [19] introduced a method to reduce the size of garbled gates from 4 to 3 ciphertexts, called *garbled row reduction (GRR)*. The idea is to exploit the freedom in choosing the output wire labels C_0, C_1 , which are not yet fixed at the time this gate is garbled. In particular, we can always make the first ciphertext G_1 equal to 0^λ . In the example above, we do so by choosing $C_0 = H(A_0, B_0)$. Since G_1 is always 0^λ , it does not need to be included in the garbled gate—the evaluator can “imagine” $G_1 = 0^\lambda$ and proceed as above.

3 Extending the KKS and WM Schemes

Kempka et al. [11], and independently Wang and Malluhi [24], give constructions of a gate-hiding garbling scheme for the class of **symmetric gates**. We now review their schemes and discuss in more detail their limitation to the class of symmetric gates.

3.1 Overview of the Constructions

In a symmetric gate, we have $g(0, 1) = g(1, 0)$. The main idea in both KKS and WM is for the false wire label A_0 and true wire label A_1 on a wire to satisfy the relationship $A_1 = A_0 + \Delta \pmod{2^\lambda}$, where Δ is a global secret constant common to all wires. Suppose the evaluator has a wire label A_a and B_b , corresponding to input combination (a, b) on some gate. Adding these wire labels mod 2^λ results in one of $\{A_0 + B_0, A_0 + B_0 + \Delta, A_0 + B_0 + 2\Delta\}$. Importantly, adding the wire labels “collapses” the two input combinations $(0, 1)$ and $(1, 0)$ to the same value.

Both the KKS and WM construction use this idea. That is, the evaluator’s first step is to add the input wire labels and use the result as input to a cryptographic hash, to obtain one of $\{H(A_0 + B_0), H(A_0 + B_0 + \Delta), H(A_0 + B_0 + 2\Delta)\}$. The corresponding result is used as a key that allow the receiver to learn the correct output wire label. The two schemes diverge significantly in their techniques at this point (specifically, WM uses polynomial interpolations), but the most important idea is this method for encoding truth values as wire labels with a global correlation by Δ .

In WM, each garbled gate requires $2\lambda + 2$ bits. In the general case of KKS, each garbled gate requires $2\lambda + 8$ bits (but see the note below about optimizations for some special cases).

Hardness assumptions. Because of the way truth values are encoded into wire labels, these schemes require a non-standard assumption. To understand why, consider that an evaluator learns one of the keys $\{H(A_0 + B_0), H(A_0 + B_0 + \Delta), H(A_0 + B_0 + 2\Delta)\}$. The security proof will have to argue that the other two keys look random. Since for all gates, the keys are related by a common secret Δ , a related-key-type assumption is used. Furthermore, since these keys are used to encrypt other wire labels in the system, which are also related by the same Δ , a circularity assumption is necessary as well. We point the reader to the KKS/WM papers for more details, and to [6] who describe the analogous situation that occurs when using Free-XOR.

Optimizations for input gates. The authors of KKS show further how to garble gates at the *input level* of the circuit for just $\lambda + 8$ bits, exploiting the extra freedom available for choosing input wire labels. In this work we focus only on the general case of *internal* gates of the circuit, whose input labels will be already fixed by the time the gate is being garbled. We justify this choice with the observation that the number of input wires is typically an extremely small fraction of the total wires in a circuit (e.g., the SHA-256 circuit has 256 inputs but over 130,000 gates [7]), so these optimizations have a relatively small effect. Certainly the difference between internal gates costing 3λ vs. 2λ bits is significantly more important.

3.2 The Limitation to Symmetric Gates, and How to Overcome It

As mentioned in Sect. 1.1, the KKS and WM schemes can garble only **symmetric gates**—those g where $g(0, 1) = g(1, 0)$. In contrast, our new constructions can garble any gate except for the two constant gates $(x, y) \mapsto 0$ and $(x, y) \mapsto 1$.

Is there a trivial modification to these schemes that avoids this limitation? We argue that the limitation to $\mathcal{G}_{\text{symm}}$ is *inherent* to these schemes, and cannot be avoided for free.

Consider an asymmetric gate $g(a, b) = a \wedge \bar{b}$ above. In order to express this gate in terms of a symmetric gate, one needs to incorporate the logic of a NOT-gate somehow. But looking closely at the choice of wire labels in KKS/WM, it seems that a NOT-gate can never be free. In particular, the wire labels satisfy $A_1 = A_0 + \Delta \pmod{2^\lambda}$ for a global Δ , and this relationship between wire labels is **not symmetric!** Contrast this with other garbling paradigms:

- In free-XOR [15] and its derivatives, the wire labels satisfy $A_1 = A_0 \oplus \Delta$, where \oplus denotes bitwise XOR. This relation *is* symmetric, so $A_0 = A_1 \oplus \Delta$ as well.
- In textbook Yao, the GRR3 scheme of [19], and the GRR2 scheme of [21], wire labels are unconstrained. A vacuous relation between wire labels is obviously symmetric.

In both of these cases, one can implement a NOT gate for free (obviously) by simply having the garbler change which label he/she considers as the false one.⁵

With wire labels in the KKS/WM paradigm, however, the two wire labels simply have their truth values “baked-in”, in a fundamental way. Their truth values cannot be swapped simply by the garbler changing his/her internal perspective.

The garbler could consider $(-\Delta \pmod{2^\lambda})$ to be the *local* wire-label-difference, just for this gate. But this would invert the truth value on *both* input wires, leaving the resulting gate functionality still symmetric. To realize a non-symmetric gates it must be necessary to negate only one of the two input wires.

Supporting non-symmetric gates via unary gates. We observe that every gate $g \in \mathcal{G}_{\text{all}}$ can be expressed as $g(a, b) = f(h(a), b)$ where f is symmetric and h is a unary gate. For example, a non-symmetric gate like $g(a, b) = a \wedge \bar{b}$ can be written using DeMorgan’s laws as $g(a, b) = \text{NOR}(\text{NOT}(a), b)$. Hence, an obvious way to extend KKS/WM beyond $\mathcal{G}_{\text{symm}}$ -gates is to incorporate unary gates. In the full version we show a simple way to garble unary gates—in a way that hides the choice of the unary gate and is compatible with KKS/WM wire labels—using only λ bits per gate, 2 additional calls to H for the garbler, 1 additional call to H for the evaluator. Now every \mathcal{G}_{all} gate can be garbled with a gadget of the form $f(h(a), b)$, costing $3\lambda + O(1)$ bits.

⁵ This fact is explicitly mentioned in [7].

4 Construction Based on Polynomial Interpolation

We review a garbling scheme of Pinkas et al. [21] (hereafter PSSW), that results in garbled gates of $2\lambda + 4$ bits. The scheme uses different garbling/evaluation approaches depending on whether the gate has even/odd parity (the parity of a gate is even if its truth table contains an even number of 1s). We then show that a simple modification allows the scheme to be gate-hiding for all $\mathcal{G}_{\text{non-const}}$ -gates (i.e., gates of either parity).

4.1 Overview of PSSW Garbling Scheme

The PSSW approach is based on polynomial interpolation. We describe only the method they propose for odd-parity gates, since that is the method we will extend to the even-parity case as well.

Suppose a gate has input wire labels (A_0, A_1) and (B_0, B_1) . Here, the subscripts will denote the **color bits** (see Sect. 2.4) of the wires. This means that the evaluator will be able to behave differently depending on the subscripts of the input wire labels he/she has.

For each input combination, we define an associated “key” value:

$$\begin{aligned} K_1 &= H(A_0, B_0); & K_3 &= H(A_1, B_0); \\ K_2 &= H(A_0, B_1); & K_4 &= H(A_1, B_1). \end{aligned}$$

The evaluator learns only one combination of input wire labels, and hence learns one of the K_i values (and knows the subscript of this K_i value, as it is determined by the input wires’ color bits). The property we require of H is that the other three K_i values look random.

For an AND gate, we want to arrange things so that learning K_i allows the evaluator to learn the corresponding output wire label of the gate. For instance, depending on the color bits, we might need to arrange for the evaluator to learn the following:

$$\begin{array}{ll} \text{knowing } K_1 \text{ lets you learn } C_0 & \text{knowing } K_3 \text{ lets you learn } C_0 \\ \text{knowing } K_2 \text{ lets you learn } C_1 & \text{knowing } K_4 \text{ lets you learn } C_0 \end{array}$$

For this example, the garbler can proceed as follows. First, use polynomial interpolation to find the unique degree-2 polynomial P passing through the points $\{(1, K_1), (3, K_3), (4, K_4)\}$. These are the cases for which the evaluator should learn C_0 . Here we take the K_i values to live in \mathbb{Z}_p for some prime p , and P is a polynomial over \mathbb{Z}_p . In practice, one would instead use a finite field $GF(2^\lambda)$, but a prime field makes the notation a little simpler.

Now the garbler finds the unique degree-2 polynomial Q passing through the points $\{(2, K_2), (5, P(5)), (6, P(6))\}$. Hence, we have:

- interpolating a polynomial through $(1, K_1), (5, P(5)), (6, P(6))$ results in P
- interpolating a polynomial through $(2, K_2), (5, P(5)), (6, P(6))$ results in Q

- interpolating a polynomial through $(3, K_3), (5, P(5)), (6, P(6))$ results in P
- interpolating a polynomial through $(4, K_4), (5, P(5)), (6, P(6))$ results in P

This suggests to make the garbled gate consist of values $P(5), P(6)$ and to set $P(0)$ and $Q(0)$ to be the false/true output wire labels. The evaluator will compute K_i , interpolate a polynomial using the garbled gate values, and then evaluate that polynomial at point 0.

Dealing with color bits. Recall that each wire label is associated with an additional color bit. The garbled gate must also contain information that tells the evaluator the color bit of the output wire label.

To do so, we interpret H as a function of the form $H : \{0, 1\}^* \rightarrow \{0, 1\}^{\lambda+1}$, and write $K_i \parallel \kappa_i = H(A, B)$, where $K_i \in \{0, 1\}^\lambda$ and $\kappa_i \in \{0, 1\}$. The garbler can choose random color bits for the output wire labels. Then for each i , he/she can encrypt the color bit of the appropriate output wire label, using κ_i as a one-time pad. This adds an additional 4 bits to the garbled gate. The evaluator will choose which 1-bit ciphertext to decrypt based on the color bits of the input wires.

4.2 Modification to Support Both Even/Odd Gates

Our main contribution in this section is to simply point out that the PSSW technique for odd gates can be extended to work for even gates as well.

We keep the same evaluation algorithm as in PSSW. Recall that the evaluator interpolates a polynomial through points $P(5)$ and $P(6)$ (which comprise the garbled gate information) and evaluates that polynomial at point 0 to obtain the output wire label. In order for this same evaluation procedure to work for an even gate, it suffices to arrange the polynomials so that **2 of the points are on P and 2 are on Q** . As an example, suppose we want K_1, K_2 to lie on P and K_3, K_4 to lie on Q . Then we require polynomials that satisfy:

$$\left\{ \begin{array}{l} P(1) = K_1 \\ P(2) = K_2 \\ Q(3) = K_3 \\ Q(4) = K_4 \\ P(5) - Q(5) = 0 \\ P(6) - Q(6) = 0 \end{array} \right\} \iff \begin{bmatrix} 1^0 & 1^1 & 1^2 & & & \\ 2^0 & 2^1 & 2^2 & & & \\ & & & 3^0 & 3^1 & 3^2 \\ & & & 4^0 & 4^1 & 4^2 \\ 5^0 & 5^1 & 5^2 & -5^0 & -5^1 & -5^2 \\ 6^0 & 6^1 & 6^2 & -6^0 & -6^1 & -6^2 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ q_0 \\ q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \\ K_4 \\ 0 \\ 0 \end{bmatrix}$$

where we write $P(x) = p_0 + p_1x + p_2x^2$ and $Q(x) = q_0 + q_1x + q_2x^2$.

Note that the K_i values are fixed (the generator has no control over them, since the input wire labels are already fixed). Hence, the generator will compute the K_i values and solve for polynomials P and Q using the above linear equation. Clearly it suffices for the 6×6 matrix to be invertible.

In the general case, the garbler will consider the following 6×6 matrix:

$$\begin{bmatrix} \tau_{00}1^0 & \tau_{00}1^1 & \tau_{00}1^2 & \overline{\tau_{00}}1^0 & \overline{\tau_{00}}1^1 & \overline{\tau_{00}}1^2 \\ \tau_{01}2^0 & \tau_{01}2^1 & \tau_{01}2^2 & \overline{\tau_{01}}2^0 & \overline{\tau_{01}}2^1 & \overline{\tau_{01}}2^2 \\ \tau_{10}3^0 & \tau_{10}3^1 & \tau_{10}3^2 & \overline{\tau_{10}}3^0 & \overline{\tau_{10}}3^1 & \overline{\tau_{10}}3^2 \\ \tau_{11}4^0 & \tau_{11}4^1 & \tau_{11}4^2 & \overline{\tau_{11}}4^0 & \overline{\tau_{11}}4^1 & \overline{\tau_{11}}4^2 \\ 5^0 & 5^1 & 5^2 & -5^0 & -5^1 & -5^2 \\ 6^0 & 6^1 & 6^2 & -6^0 & -6^1 & -6^2 \end{bmatrix} \tag{1}$$

for some $(\tau_{00}, \tau_{01}, \tau_{10}, \tau_{11}) \in \{0, 1\}^4 \setminus \{0000, 1111\}$ (corresponding to a non-constant gate). One can verify by hand that the above matrix is invertible in all cases. This is also true in $GF(2^\lambda)$ if we replace integers $\{1, \dots, 6\}$ with finite field elements whose representations in hex are $\{0x1, \dots, 0x6\}$.

```

Garble( $1^\lambda, f$ ):
  for  $i \in \text{inputs}(f)$ :
     $\sigma_i \leftarrow \{0, 1\}$  // color bit representing false on this wire
     $W_i^0 \leftarrow (\{0, 1\}^\lambda \parallel \sigma_i)$ 
     $W_i^1 \leftarrow (\{0, 1\}^\lambda \parallel \overline{\sigma_i})$ 
   $e := \left( (W_i^0, W_i^1) \right)_{i \in \text{inputs}(f)}$ 

  for  $i \in \text{gates}(f)$ :
     $(W_i^0, W_i^1, F_i) \leftarrow \text{GbGate}(\text{type}(i), i; W_{\text{left}(i)}^0, W_{\text{left}(i)}^1, W_{\text{right}(i)}^0, W_{\text{right}(i)}^1)$ 
   $F = (F_i)_{i \in \text{gates}(f)}$ 

  for  $i \in \text{outputs}(f)$ :
     $d_i^0 := H(i, \text{OUT}, \text{lsb}(W_i^0); W_i^0, 0^\lambda)$ 
     $d_i^1 := H(i, \text{OUT}, \text{lsb}(W_i^1); W_i^1, 0^\lambda)$ 
   $d := \left( (d_i^0, d_i^1) \right)_{i \in \text{outputs}(f)}$ 
  return  $(F, e, d)$ 

Encode( $e, x$ ):
  parse  $e$  as  $\left( (W_i^0, W_i^1) \right)_{i \in \text{inputs}(f)}$ 
  return  $(W_i^{x_i})_{i \in \text{inputs}(f)}$ 

Decode( $d, Y$ ):
  parse  $d$  as  $\left( (d_i^0, d_i^1) \right)_{i \in \text{outputs}(f)}$ 
  parse  $Y$  as  $(Y_i)_{i \in [|Y|]}$ 

  for  $i \in [|Y|]$ :
    if  $Y_i = d_i^0$  then  $y_i := 0$ 
    else if  $Y_i = d_i^1$  then  $y_i := 1$ 
    else return  $\perp$ 
  return  $y = y_1 \cdots y_{|Y|}$ 

Eval( $F, X$ ):
  parse  $X$  as  $(W_i^*)_{i \in \text{inputs}(f)}$ 
  parse  $F$  as  $(F_i)_{i \in \text{gates}(f)}$ 
  for  $i \in \text{gates}(f)$ :
     $W_i^* \leftarrow \text{EvGate}(i; W_{\text{left}(i)}^*, W_{\text{right}(i)}^*, F_i)$ 
  return  $\left( H(i, \text{OUT}, \text{lsb}(W_i^*); W_i^*, 0^\lambda) \right)_{i \in \text{outputs}(f)}$ 

```

Fig. 2. Generic template for garbling scheme. **GbGate** and **EvGate** subroutines to be specified later. Notation **type**(i) (used in **Garble**) refers to the gate functionality of the gate with index i (see Sect. 2.1).

4.3 Details

In Fig. 2 we give a generic template for a garbling scheme. Both of our new constructions will instantiate this template. In the template for the garbler, we use W_i^b as the wire label representing truth value b on wire i , and σ_i to denote the color bit on W_i^0 . For the evaluator, we use W_i^* as the “active” wire label

```

// INPUT: gate with functionality  $g$  and index  $idx$ ;
           left wire labels  $(A_0, A_1)$ ; right wire labels  $(B_0, B_1)$ 
// OUTPUT: output wire labels  $(C_0, C_1)$ ; garbled gate information

GbGate( $g, idx; A_0, A_1, B_0, B_1$ ):
   $\sigma_A := \text{lsb}(A_0)$ 
   $\sigma_B := \text{lsb}(B_0)$ 
   $\sigma_C \leftarrow \{0, 1\}$ 

  for  $a, b \in \{0, 1\}^2$ :
     $\tau_{ab} := g(\sigma_A \oplus a, \sigma_B \oplus b)$  // each combination of visible color bits:
     $K_{ab} \parallel \kappa_{ab} := H(idx, a, b; A_{\sigma_A \oplus a}, B_{\sigma_B \oplus b})$  // gate output truth value

  
$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ q_0 \\ q_1 \\ q_2 \end{bmatrix} := \begin{bmatrix} \tau_{00} & \tau_{00} \cdot 0x1 & \tau_{00} \cdot 0x1^2 & \overline{\tau_{00}} & \overline{\tau_{00}} \cdot 0x1 & \overline{\tau_{00}} \cdot 0x1^2 \\ \tau_{01} & \tau_{01} \cdot 0x2 & \tau_{01} \cdot 0x2^2 & \overline{\tau_{01}} & \overline{\tau_{01}} \cdot 0x2 & \overline{\tau_{01}} \cdot 0x2^2 \\ \tau_{10} & \tau_{10} \cdot 0x3 & \tau_{10} \cdot 0x3^2 & \overline{\tau_{10}} & \overline{\tau_{10}} \cdot 0x3 & \overline{\tau_{10}} \cdot 0x3^2 \\ \tau_{11} & \tau_{11} \cdot 0x4 & \tau_{11} \cdot 0x4^2 & \overline{\tau_{11}} & \overline{\tau_{11}} \cdot 0x4 & \overline{\tau_{11}} \cdot 0x4^2 \\ 1 & 0x5 & 0x5^2 & 1 & 0x5 & 0x5^2 \\ 1 & 0x6 & 0x6^2 & 1 & 0x6 & 0x6^2 \end{bmatrix}^{-1} \begin{bmatrix} K_{00} \\ K_{01} \\ K_{10} \\ K_{11} \\ 0 \\ 0 \end{bmatrix} // GF(2^\lambda)$$


   $C_0 := p_0 \parallel \sigma_C$ 
   $C_1 := q_0 \parallel \overline{\sigma_C}$ 

   $G := p_0 + p_1 \cdot 0x5 + p_2 \cdot 0x5^2$  //  $GF(2^\lambda)$ 
   $G' := p_0 + p_1 \cdot 0x6 + p_2 \cdot 0x6^2$  //  $GF(2^\lambda)$ 

  for  $a, b \in \{0, 1\}^2$ :
    set  $c_{ab} := \kappa_{ab} \oplus \sigma_C \oplus \tau_{ab}$  // encrypted output wire label color bit

  return  $(C_0, C_1, (G, G', c_{00}, c_{01}, c_{10}, c_{11}))$ 

// INPUT: gate index  $idx$ ; left wire label  $A^*$ ; right wire label  $B^*$ ; garbled gate info
// OUTPUT: output wire label  $C^*$ 

EvGate( $idx; A^*, B^*, (G, G', c_{00}, c_{01}, c_{10}, c_{11})$ ):
   $\chi_A := \text{lsb}(A^*)$ 
   $\chi_B := \text{lsb}(B^*)$ 
   $K^* \parallel \kappa^* := H(idx, \chi_A, \chi_B; A^*, B^*)$ 
   $R :=$  unique degree-2 polynomial in  $GF(2^\lambda)$  passing through
            $\{(2\chi_A + \chi_B + 1, K^*), (5, G), (6, G')\}$ 
   $\chi_C := c_{\chi_A, \chi_B} \oplus \kappa^*$ 
  return  $C^* = R(0) \parallel \chi_C$ 

```

Fig. 3. Our gate-hiding garbling scheme based on polynomial evaluation.

(representing unknown truth value) on wire i , and χ_i to denote its color bit (known to the evaluator).

We give the specific procedure for garbling & evaluating gates of this scheme in Fig. 3. Departing from the (simplified) discussion above, we include the color bits in the input to H to ensure that all “nonce” arguments to H are globally unique (as required by the dual-key hash definition); e.g., $K_{ab} \parallel \kappa_{ab} := H(\text{id}x, a, b; \dots)$.

The scheme is written to involve an explicit matrix inverse, as in Eq. 1 above. In practice, there are only $16 - 2 = 14$ possible matrices (one for each non-constant setting of the τ_{ab} bits), and their inverses would all be easily hard-coded into a lookup table.

In the full version we prove the following

Theorem 1. *The construction in Figs. 2 and 3 satisfies the gate-hiding privacy, obliviousness (both with respect to $\mathcal{G}_{\text{non-const}}$), and authenticity properties (Sect. 2.2), if H is a dual-key hash.*

5 Construction that Avoids Polynomial Interpolation

5.1 Overview of GLNP Garbling Scheme

In [7], Gueron et al. describe a different way to garble gates (in their case, odd-parity gates only) at a cost of 2 ciphertexts. While the construction of [21] involves polynomial interpolation, the construction of [7] involves only cheap XOR operations, making it preferable both in performance and ease of implementation.

The main idea is to start with the classical point-and-permute Yao scheme, in which the garbled gate consists of 4 ciphertexts. Suppose a gate has input wire labels (A_0, A_1) and (B_0, B_1) , and output wire labels (C_0, C_1) . As before, the subscripts correspond to the visible color bits. Consider the following example odd-parity gate in the textbook Yao scheme:

$$\begin{aligned} G_1 &= H(A_0, B_0) \oplus C_0 & G_3 &= H(A_1, B_0) \oplus C_0 \\ G_2 &= H(A_0, B_1) \oplus C_1 & G_4 &= H(A_1, B_1) \oplus C_0 \end{aligned}$$

The high level idea is to exploit the two degrees of freedom in the choice of output labels C_0 and C_1 , which are not yet chosen at the time this gate will be garbled. The first step is to apply the GRR3 row reduction of [19], setting $G_1 = 0^\lambda$. In this example, we can do so by choosing C_0 in a special way: $C_0 = H(A_0, B_0)$.

After fixing one of the output wire labels, the next step is to fix the other output label so that $G_2 \oplus G_3 \oplus G_4 = 0^\lambda$. In this example, we can do so by choosing $C_1 = H(A_0, B_1) \oplus H(A_1, B_0) \oplus H(A_1, B_1)$.

By choosing C_0 and C_1 in this way, the garbler has guaranteed that $G_1 = G_2 \oplus G_3 \oplus G_4 = 0^\lambda$. Hence, G_1 does not need to be sent (it is always all zeroes), and neither does G_4 (it can always be reconstructed by the receiver as $G_2 \oplus G_3$). In this way, only two ciphertexts actually need to be sent.

The problem of even-parity gates. One can check that the GLNP technique works for *any* odd-parity gate, but does **not** work when the gate has even parity. We illustrate with an example. Consider the following classical-Yao garbled gate for an even-parity truth table:

$$\begin{aligned} G_1 &= H(A_0, B_0) \oplus C_0 & G_3 &= H(A_1, B_0) \oplus C_1 \\ G_2 &= H(A_0, B_1) \oplus C_1 & G_4 &= H(A_1, B_1) \oplus C_0 \end{aligned}$$

To achieve $G_1 = 0^\lambda$ we must set $C_0 = H(A_0, B_0)$ as before. But now observe that $G_2 \oplus G_3 \oplus G_4 = H(A_0, B_1) \oplus H(A_1, B_0) \oplus H(A_1, B_1) \oplus C_0$, a value that is already fixed! Because of the even parity of the gate, the C_1 terms cancel out in this expression. There is no way to choose C_1 so that $G_2 \oplus G_3 \oplus G_4 = 0^\lambda$ as before.

5.2 Our Construction

Let us look a little more abstractly at the GLNP scheme. The evaluator computes K (the hash of the two wire labels) and then obtains the final wire label as $K \oplus \alpha G \oplus \beta G'$, where G and G' are the two “ciphertexts” in the garbled gate, and α and β are bits that depend on color bits of the wire labels. The mapping between color bits and α, β coefficients is *fixed* for the entire scheme.

Our approach is to *randomize* and *partially hide* this mapping of color bits to α, β coefficients. In our scheme, evaluation works as follows:

- The evaluator hashes the input wire labels to compute a key K .
- The evaluator uses K to decrypt a 2-bit ciphertext containing $\alpha||\beta$. There are four such 2-bit ciphertexts, arranged according to color bits. The evaluator uses the color bits of the input labels to determine which of these 2-bit ciphertexts to decrypt.
- Having obtained the appropriate α, β , the evaluator computes the output wire label as $K \oplus \alpha G \oplus \beta G'$.

So each garbled gate consists of the following:

- G and G' (2λ bits)
- four 2-bit ciphertexts that encrypt α, β values (8 bits total)
- four 1-bit ciphertexts that encrypt the color bit of the output wire label (4 bits total)

The real power of the scheme comes from the *indirection* in conveying the evaluator’s final linear combination. The evaluator uses his/her color bits to decrypt a constant-sized ciphertext, which tells him/her what linear combination to finally apply. A similar kind of indirection also appears in the construction of [11], where they use it to circumvent a lower bound for “linear garbling” from [26] (the model for the lower bound implicitly assumes a direct, fixed correspondence between color bits and the evaluator’s final linear combination).

Now let us consider how the garbler can arrange for all of this to happen. Let C_0 and C_1 denote the false/true output wire labels (yet to be determined).

Let K_1, \dots, K_4 be the four possible input hashes, as before. Let α_i, β_i denote the coefficients that the evaluator will use when he/she has K_i . Below is an example of the correctness conditions required for an example gate:

$$\begin{aligned}
 C_0 &= K_1 \oplus \alpha_1 G \oplus \beta_1 G' \\
 C_0 &= K_2 \oplus \alpha_2 G \oplus \beta_2 G' \\
 C_1 &= K_3 \oplus \alpha_3 G \oplus \beta_3 G' \\
 C_0 &= K_4 \oplus \alpha_4 G \oplus \beta_4 G'
 \end{aligned}
 \iff
 \begin{bmatrix} K_1 \\ K_2 \\ K_3 \\ K_4 \end{bmatrix}
 =
 \begin{bmatrix} 1 & 0 & \alpha_1 & \beta_1 \\ 1 & 0 & \alpha_2 & \beta_2 \\ 0 & 1 & \alpha_3 & \beta_3 \\ 1 & 0 & \alpha_4 & \beta_4 \end{bmatrix}
 \begin{bmatrix} C_0 \\ C_1 \\ G \\ G' \end{bmatrix}
 \tag{2}$$

We let the garbler choose $\{\alpha_i, \beta_i\}$ values uniformly subject to the matrix in Eq. 2 being invertible. Importantly, for different gate types, *this is a different distribution* over the α_i, β_i values! In particular, when the gate has odd parity (i.e., the parity of the first column is odd), there are 96 ways to choose α_i, β_i coefficients to make the matrix invertible. When the gate has even parity, there are 104 ways. Beyond that, the distributions are different even for different gates of the same parity. Below we discuss in more detail how this difference affects security.

In summary, the garbler computes K_1, \dots, K_4 (these are fixed by the choice of the input wire labels), chooses random α_i, β_i values that make the appropriate matrix invertible, and finally solves for consistent C_0, C_1, G, G' according to Eq. 2. The C_0, C_1 values will be the output wire labels and (G, G') will be the garbled gate (along with the 12 bits of encryptions mentioned above).

Why it hides the gate type. From the evaluator’s perspective, every kind of gate is handled the same way—decrypt the correct α, β and output $K \oplus \alpha G \oplus \beta G'$.

However, the garbler’s behavior depends on the choice of gate. In particular, he/she uses a different distribution over the α_i, β_i values for different gates. We must ensure that this difference is not noticeable to the evaluator. The key point is that the evaluator sees **only a single α_i, β_i pair** while the other coefficients remain encrypted.⁶ Furthermore, all possible garbling distributions have the property that for every i , the marginal distribution of (α_i, β_i) is uniform. Hence, the evaluator sees only a uniform α_i, β_i , regardless of the gate type.

To see why this is true, take any vector $v \in \{0, 1\}^4 \setminus \{0000, 1111\}$ and consider any invertible matrix (over \mathbb{Z}_2) of the following form:

$$\begin{bmatrix} v_1 & \overline{v_1} & \alpha_1 & \beta_1 \\ v_2 & \overline{v_2} & \alpha_2 & \beta_2 \\ v_3 & \overline{v_3} & \alpha_3 & \beta_3 \\ v_4 & \overline{v_4} & \alpha_4 & \beta_4 \end{bmatrix}
 \tag{3}$$

Note that flipping every bit in the 3rd column is an elementary matrix operation, since the first two columns sum to the all-ones vector. Hence this modification has no effect on the determinant. This modification is also invertible. Thus, for

⁶ If all the coefficients were to be made public, then we are back in the original situation of [7], and leaking the parity of the gate seems inevitable for this choice of evaluation equation.

any i , we have an 1-to-1 correspondence between the set of invertible matrices with (α_i, β_i) and those with $(\overline{\alpha}_i, \beta_i)$. Of course, the same can be said for the mappings that flip every bit in the 4th column, or in both the 3rd and 4th columns. Hence, after fixing v , the number of ways to complete the matrix in an invertible way does not depend on the choice of a particular (α_i, β_i) .

```

// INPUT: gate with functionality g and index idx;
//         left wire labels (A0, A1); right wire labels (B0, B1)
// OUTPUT: output wire labels (C0, C1); garbled gate information

GbGate(g; A0, A1, B0, B1):
   $\sigma_A := \text{lsb}(A_0)$ 
   $\sigma_B := \text{lsb}(B_0)$ 
   $\sigma_C \leftarrow \{0, 1\}$ 

  for  $a, b \in \{0, 1\}^2$ : // each combination of visible color bits:
     $\tau_{ab} := g(\sigma_A \oplus a, \sigma_B \oplus b)$  // gate output truth value
     $K_{ab} \parallel \kappa_{ab} \parallel \widehat{\kappa}_{ab} := H(\text{idx}, a, b; A_{\sigma_A \oplus a}, B_{\sigma_B \oplus b})$ 

  sample  $\{\alpha_{ab}, \beta_{ab} \mid a, b \in \{0, 1\}\}$  uniformly s.t. the matrix below is invertible

  
$$\begin{bmatrix} \tilde{C}_0 \\ \tilde{C}_1 \\ G \\ G' \end{bmatrix} := \begin{bmatrix} \overline{\tau}_{00} & \tau_{00} & \alpha_{00} & \beta_{00} \\ \overline{\tau}_{01} & \tau_{01} & \alpha_{01} & \beta_{01} \\ \overline{\tau}_{10} & \tau_{10} & \alpha_{10} & \beta_{10} \\ \overline{\tau}_{11} & \tau_{11} & \alpha_{11} & \beta_{11} \end{bmatrix}^{-1} \times \begin{bmatrix} K_{00} \\ K_{01} \\ K_{10} \\ K_{11} \end{bmatrix}$$


   $C_0 := \tilde{C}_0 \parallel \sigma_C$ 
   $C_1 := \tilde{C}_1 \parallel \overline{\sigma_C}$ 

  for  $a, b \in \{0, 1\}^2$ :
     $c_{ab} := \kappa_{ab} \oplus \sigma_C \oplus \tau_{ab}$  // encrypted output wire label color bit
     $d_{ab} := \widehat{\kappa}_{ab} \oplus (\alpha_{ab} \parallel \beta_{ab})$  // encrypted  $\alpha, \beta$  coefficients

  return  $(C_0, C_1, (G, G', c_{00}, \dots, c_{11}, d_{00}, \dots, d_{11}))$ 

// INPUT: gate index idx; left wire label A*; right wire label B*; garbled gate info
// OUTPUT: output wire label C*

EvGate(A*, B*, (G, G', c00, ..., c11, d00, ..., d11)):
   $\chi_A := \text{lsb}(A^*)$ 
   $\chi_B := \text{lsb}(B^*)$ 
   $K^* \parallel \kappa^* \parallel \widehat{\kappa}^* := H(\text{idx}, \chi_A, \chi_B; A^*, B^*)$ 
   $\alpha^* \parallel \beta^* := d_{\chi_A, \chi_B} \oplus \widehat{\kappa}^*$ 
   $C := K^* \oplus \alpha^* G \oplus \beta^* G'$ 
   $\chi_C := c_{\chi_A, \chi_B} \oplus \kappa^*$ 
  return  $C \parallel \chi_C$ 

```

Fig. 4. Our gate-hiding garbling scheme that uses only XOR

More details about evaluation. Similar to the previous construction, we assume a cryptographic hash function of the form $H : \{0, 1\}^* \rightarrow \{0, 1\}^{\lambda+3}$. When the evaluator has input wire labels A and B , he/she evaluates $K \parallel \kappa \parallel \widehat{\kappa} \leftarrow H(A, B)$, where $K \in \{0, 1\}^\lambda$, $\kappa \in \{0, 1\}$, and $\widehat{\kappa} \in \{0, 1\}^2$. Then:

- $\widehat{\kappa}$ is used as a one-time pad key to decrypt the α, β coefficients.
- K is used to compute the output wire label as $K \oplus \alpha G \oplus \beta G'$.
- κ is used as a one-time pad key to decrypt the output wire label's color bit.

As in the previous scheme, some of the garbler's computations can be hard-coded into lookup tables in an implementation. Each possible value of $\tau_{00} \cdots \tau_{11}$ (14 choices) can index a list of valid α_{ab}, β_{ab} values along with the inverse of the appropriate matrix from Eq. 3. These lookup tables will clearly be more extensive for this scheme than for the previous one, but overall have reasonable constant size.

5.3 Formal Details

In Fig. 4 we give the formal details of the construction. It follows the high-level discussion above. In the full version we prove the following:

Theorem 2. *The construction in Figs. 2 and 4 satisfies the gate-hiding privacy, obliviousness (both with respect to $\mathcal{G}_{\text{non-const}}$), and authenticity properties (Sect. 2.2), if H is a dual-key hash.*

Saving 4 bits. As described, this scheme requires $2\lambda + 12$ bits per gate. In the full version we show how to modify the construction to require only $2\lambda + 8$ bits.

References

1. Abadi, M., Feigenbaum, J.: Secure circuit evaluation. *J. Cryptol.* **2**(1), 1–12 (1990)
2. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC, pp. 503–513. ACM Press, May 1990
3. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: 2013 IEEE Symposium on Security and Privacy, pp. 478–492. IEEE Computer Society Press, May 2013
4. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 784–796. ACM Press, October 2012
5. Bellare, M., Lysyanskaya, A.: Symmetric and dual PRFs from standard assumptions: a generic validation of an HMAC assumption. *Cryptology ePrint Archive, Report 2015/1198* (2015). <http://eprint.iacr.org/2015/1198>
6. Choi, S.G., Katz, J., Kumaresan, R., Zhou, H.-S.: On the security of the “Free-XOR” technique. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 39–53. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_3
7. Gueron, S., Lindell, Y., Nof, A., Pinkas, B.: Fast garbling of circuits under standard assumptions. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 567–578. ACM Press, October 2015

8. Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. Cryptology ePrint Archive, Report 2017/798 (2017). <http://eprint.iacr.org/2017/798>
9. Henecka, W., Schneider, T.: Memory efficient secure function evaluation. <https://code.google.com/p/me-sfe/>
10. Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_30
11. Kempka, C., Kikuchi, R., Suzuki, K.: How to circumvent the two-ciphertext lower bound for linear garbling schemes. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 967–997. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_32
12. Kennedy, W.S., Kolesnikov, V., Wilfong, G.: Overlaying circuit clauses for secure computation. Cryptology ePrint Archive, Report 2016/685 (2016). <http://eprint.iacr.org/2016/685>
13. Kiss, Á., Schneider, T.: Valiant’s universal circuit is practical. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 699–728. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_27
14. Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: flexible garbling for XOR gates that beats free-XOR. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 440–457. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_25
15. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
16. Lindell, Y., Pinkas, B.: A proof of security of Yao’s protocol for two-party computation. *J. Cryptol.* **22**(2), 161–188 (2009)
17. Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant’s universal circuit: improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017 (2016). <http://eprint.iacr.org/2016/017>
18. Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_33
19. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce, pp. 129–139. ACM, New York (1999)
20. Paus, A., Sadeghi, A.-R., Schneider, T.: Practical secure evaluation of semi-private functions. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 89–106. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_6
21. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_15
22. Tillich, S., Smart, N.: Circuits of basic functions suitable for MPC and FHE. <http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/>

23. Valiant, L.G.: Universal circuits (preliminary report). In: Chandra, A.K., Wotschke, D., Friedman, E.P., Harrison, M.A. (eds.) Proceedings of the 8th Annual ACM Symposium on Theory of Computing, Hershey, Pennsylvania, USA, pp. 196–203. ACM, 3–5 May 1976
24. Wang, Y., Malluhi, Q.: Reducing garbled circuit size while preserving circuit gate privacy. Cryptology ePrint Archive, Report 2017/041 (2017). <http://eprint.iacr.org/2017/041>
25. Yao, A.C.-C.: Protocols for secure computations (extended abstract). In: 23rd FOCS, pp. 160–164. IEEE Computer Society Press, November 1982
26. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8

Recovering Short Generators of Principal Fractional Ideals in Cyclotomic Fields of Conductor $p^\alpha q^\beta$

Patrick Holzer, Thomas Wunderer^(✉), and Johannes A. Buchmann

TU Darmstadt, Darmstadt, Germany
patrick.holzer@stud.tu-darmstadt.de,
{twunderer,buchmann}@cdc.informatik.tu-darmstadt.de

Abstract. Several recent cryptographic constructions – including a public key encryption scheme, a fully homomorphic encryption scheme, and a candidate multilinear map construction – rely on the hardness of the *short generator principal ideal problem* (SG-PIP): given a \mathbb{Z} -basis of some principal (fractional) ideal in an algebraic number field that is guaranteed to have an exceptionally short generator, find a shortest generator of the principal ideal. The folklore approach to this problem is to first, recover some arbitrary generator of the ideal, which is known as the *principal ideal problem* (PIP) and second, solve a bounded distance decoding (BDD) problem in the *log-unit lattice* to transform this arbitrary generator into a shortest one. The PIP can be solved in polynomial time on *quantum* computers for arbitrary number fields under the *generalized Riemann hypothesis* due to Biasse and Song. Cramer et al. showed, based on the work of Campbell et al., that the second problem can be solved in polynomial time on *classical* computers for *cyclotomic fields of prime-power conductor*.

In this work, we extend the work of Cramer et al. to cyclotomic fields $K = \mathbb{Q}(\xi_m)$ of conductor $m = p^\alpha q^\beta$, where p, q are distinct odd primes.

In more detail, we show that the BDD problem in the log-unit lattice can be solved in classical polynomial time (with quantum polynomial time precomputation) under some sufficient conditions, if (p, q) is an (α, β) -generator prime pair, a new notion introduced in this work.

Keywords: Lattice-based cryptography · Principal ideal lattices
SG-PIP · Cryptanalysis

1 Introduction

Over the past decade, lattice-based cryptography [23] has emerged as one of the most promising candidates for post-quantum cryptography [18]. The security of lattice-based schemes relies on the hardness of lattice problems such as

T. Wunderer—This research was supported by the DFG as part of project P1 within the CRC 1119 CROSSING.

finding a shortest non-zero vector of a lattice. In order to boost the efficiency or achieve additional functionality, more structured lattices have been taken into consideration, for example lattices induced by (principal) fractional ideals in algebraic number fields, called *ideal lattices* [16, 17]. Some recent cryptographic constructions – including a public key encryption scheme [6], a fully homomorphic encryption scheme [26], and a candidate multilinear map construction [11] – rely on the hardness of the *short generator principal ideal problem (SG-PIP)* [8]: Given a \mathbb{Z} -basis of a principal fractional ideal \mathfrak{a} in some algebraic number field K that is guaranteed to have an exceptionally short generator, find a shortest generator of \mathfrak{a} .

The folklore approach to solve this problem, as sketched by Bernstein [2] and Campbell et al. [6] is to split it into the following two problems.

1. Recover some arbitrary generator $g' \in K$ of the ideal \mathfrak{a} , which is known as the *principal ideal problem (PIP)*.
2. Transform this generator into some shortest generator. In more detail, let $g = ug'$ for some unit $u \in \mathcal{O}_K^\times$ be a shortest generator of \mathfrak{a} with respect to the logarithmic embedding. In this case it holds that $\text{Log}(g') \in \text{Log}(g) + \text{Log}(\mathcal{O}_K^\times)$, where Log denotes the logarithmic embedding. Since $\text{Log}(g)$ is short, we can therefore find $\text{Log}(g)$ (and hence g) by solving a closest vector problem in the *Dirichlet log-unit lattice* $\text{Log}(\mathcal{O}_K^\times)$.

The PIP can be solved in polynomial time on quantum computers for cyclotomic fields $K = \mathbb{Q}(\xi_m)$ of prime-power conductor $m = p^\alpha$ [4, 6, 10] and, under the *generalized Riemann hypothesis*, also for arbitrary number fields [5]. Following the sketch of Campbell et al. [6], Cramer et al. [8] proved that the second problem can be solved in classical polynomial time for cyclotomic fields $K = \mathbb{Q}(\xi_m)$ of prime-power conductor $m = p^\alpha$, under some conjecture concerning the class number h_m^+ of $K^+ = \mathbb{Q}(\xi_m + \xi_m^{-1})$. Their algorithm relies on the fact that the units $\frac{\xi_m^j - 1}{\xi_m - 1} \in \mathbb{Z}[\xi_m]^\times$ for $j \in \mathbb{Z}_m/\{\pm 1\}$ form a well suited basis of the so called *cyclotomic units*, a subgroup of finite index in the unit group $\mathcal{O}_K^\times = \mathbb{Z}[\xi_m]^\times$ in the prime-power case $m = p^\alpha$. The success of their algorithm relies on the following two facts.

1. The index of the group of cyclotomic units in $\mathbb{Z}[\xi_m]^\times$ is sufficiently small, i.e., bounded by some constant (or at least by some polynomial in $n = \varphi(m)$) if m is a prime-power.
2. The norm of the dual vectors $\text{Log}\left(\frac{\xi_m^j - 1}{\xi_m - 1}\right)^*$ for all $j \in \mathbb{Z}_m/\{\pm 1\}$ is small enough if m is a prime-power.

The proofs given in [8] heavily use that the underlying cyclotomic number fields have prime-power conductor.

In this work, we extend the work of Cramer et al. to cyclotomic number fields $K = \mathbb{Q}(\xi_m)$ of conductor $m = p^\alpha q^\beta$, where p, q are distinct odd primes. We show that in this case, under some conditions, the BDD problem in the log-unit lattice can efficiently be solved if (p, q) is an (α, β) -*generator prime pair*, a new notion

introduced in this work. We further provide experimental evidence that suggests that roughly 35% of prime pairs are (α, β) -generator prime pairs for all α and β . Combined with the results of Biasse and Song [5] our results show that under sufficient conditions, the SG-PIP can be solved in quantum polynomial time in cyclotomic number fields of composite conductor of the form $p^\alpha q^\beta$.

In consequence, we extend the quantum polynomial time key-recover attacks [6, 8] on the cryptographic schemes of [6, 11, 26] to the case of cyclotomic number fields $\mathbb{Q}(\xi_m)$ of conductor $m = p^\alpha q^\beta$ for (α, β) -generator prime pairs (p, q) .

Outline. This work is structured as follows. In Sect. 2, we provide the necessary mathematical background for this work. In Sect. 3, we sketch the algorithmic approach and sufficient success conditions presented in [2, 6, 8] to find a shortest generator of some principal fractional ideal, given an arbitrary generator. In Sect. 4, we derive sufficient conditions, under which the algorithmic approach described in the previous section is successful in the case of cyclotomic fields of conductor $m = p^\alpha q^\beta$.

2 Preliminaries

We denote $\mathbb{N} := \{1, 2, 3, \dots\}$ and $\mathbb{N}_0 := \{0, 1, 2, 3, \dots\}$. The set of primes is denoted by \mathbb{P} . We denote the real and imaginary part of a complex number $z \in \mathbb{C}$ by $\Re(z)$ and $\Im(z)$, respectively. We use the common notation “**iff**” for “if and only if”. We denote vectors by lower-case bold letters, e.g., $\mathbf{x} \in \mathbb{R}^n$, and matrices by upper-case bold letters, e.g., $\mathbf{X} \in \mathbb{R}^{n \times m}$. For $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{R}^n$ we write $(\mathbf{x}_1, \dots, \mathbf{x}_k) =: \mathbf{X} \in \mathbb{R}^{n \times k}$ for the $n \times k$ matrix \mathbf{X} whose columns are the vectors $\mathbf{x}_1, \dots, \mathbf{x}_k$. The canonical inner product and the Euclidean norm over \mathbb{R}^n are denoted by $\langle \cdot, \cdot \rangle$ and $\|\cdot\|_2$. The common rounding function is denoted by $\lfloor x \rfloor = \lfloor x + \frac{1}{2} \rfloor \in \mathbb{Z}$. For a vector $\mathbf{v} = (v_1, \dots, v_n)^T \in \mathbb{R}^n$ we define $\lfloor v \rfloor := (\lfloor v_1 \rfloor, \dots, \lfloor v_n \rfloor)^T \in \mathbb{Z}^n$ component wise.

2.1 Lattices

A **lattice** \mathcal{L} is an additive subgroup of an n -dimensional \mathbb{R} -vectorspace V such that there exists \mathbb{R} -linearly independent vectors $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$ with $\mathcal{L} = \mathbb{Z}\mathbf{v}_1 + \dots + \mathbb{Z}\mathbf{v}_k$. The vectors $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$ are called **basis** of the lattice \mathcal{L} . If $V = \mathbb{R}^n$, we write $\mathcal{L}(\mathbf{B}) := \mathbb{Z}\mathbf{b}_1 + \dots + \mathbb{Z}\mathbf{b}_k$ for the lattice whose basis is given by the columns of a matrix $\mathbf{B} \in \mathbb{R}^{n \times k}$. The **dimension** of a lattice is defined as $\dim \mathcal{L} := k$. A **full rank** lattice is a lattice with $n = k = \dim \mathcal{L}$. A **sublattice** \mathcal{L}' of \mathcal{L} is a lattice with $\mathcal{L}' \subseteq \mathcal{L}$. The **dual basis** $\mathbf{B}^* = (\mathbf{b}_1^*, \dots, \mathbf{b}_k^*) \in \mathbb{R}^{n \times k}$ of a lattice basis $\mathbf{B} \subseteq \mathbb{R}^n$ is defined as the \mathbb{R} -basis of $\text{span}_{\mathbb{R}}(\mathbf{B})$ with $\langle \mathbf{b}_i^*, \mathbf{b}_j \rangle = \delta_{i,j}$ for all $i, j \in \{1, \dots, k\}$, i.e., $\mathbf{B}^T \cdot \mathbf{B}^* = (\mathbf{B}^*)^T \cdot \mathbf{B} = \mathbf{I}_k$. It is easy to see that the unique dual basis \mathbf{B}^* is given by $\mathbf{B}^* = \mathbf{B}(\mathbf{B}^T \mathbf{B})^{-1}$.

2.2 Algebraic Number Fields

Let L be a field and $K \subseteq L$ a subfield of L . We denote the index of K in L by $[L : K] := \dim_K L$. An **algebraic number field** K is an extension field of \mathbb{Q} of finite index. For an algebraic number field K we define the (finite) group of roots of unity as $\mu(K) := \{x \in K \mid x^n = 1 \text{ for some } n \in \mathbb{N}\}$ and its **ring of integers** \mathcal{O}_K as

$$\mathcal{O}_K := \{\alpha \in K \mid \exists p \in \mathbb{Z}[X] \setminus \{0\} : p \text{ is monic and } p(\alpha) = 0\}.$$

We say $\alpha \in K$ is **integral** iff $\alpha \in \mathcal{O}_K$. Without loss of generality it is sufficient to consider $K \subseteq \mathbb{C}$ for an algebraic number field K , since there is only one algebraic closure of \mathbb{Q} up to isomorphisms, so we assume $\overline{\mathbb{Q}} \subseteq \mathbb{C}$. Note that \mathcal{O}_K is a subring of K , see for example [22, p. 7]. A **principal fractional ideal** in K is a subring of K of the form $g\mathcal{O}_K$ for some $g \in K^\times$. The **class group** $\text{Cl}_K = \mathcal{I}_K/\mathcal{P}_K$ of K is the quotient of the abelian multiplicative group of fractional ideal \mathcal{I}_K and the subgroup of principal fractional ideals \mathcal{P}_K . The **class number** h_K of an algebraic number field K is $h_K := |\text{Cl}_K| < \infty$, see [22, Sect. 3. Ideals].

2.3 Logarithmic Embedding

Let K be an algebraic number field of degree $n = [K : \mathbb{Q}]$. Moreover, let r be the number of real embeddings $\delta_1, \dots, \delta_r : K \rightarrow \mathbb{R}$ of K and s the number of non real embeddings (up to complex conjugation) $\sigma_1, \overline{\sigma_1}, \dots, \sigma_s, \overline{\sigma_s} : K \rightarrow \mathbb{C}$. Note that $n = r + 2s$ holds. In this case, we call (r, s) the *signature* of the number field K . We define the **logarithmic embedding** as

$$\begin{aligned} \text{Log} : K^\times &\rightarrow \mathbb{R}^{r+2s} \\ x &\mapsto (\log(|\delta_1(x)|), \dots, \log(|\delta_r(x)|), \log(|\sigma_1(x)|), \dots, \log(|\overline{\sigma_s}(x)|)), \end{aligned}$$

This mapping defines a group homomorphism from the multiplicative group K^\times to the additive group $\mathbb{R}^{r+2s} = \mathbb{R}^n$. If the number field K has no real embedding, i.e., $r = 0$ and therefore $n = 2s$, it is sufficient to use the **reduced logarithmic embedding** of K^\times :

$$\text{Log}_r(x) := (\log(|\sigma_1(x)|), \dots, \log(|\sigma_s(x)|)) \in \mathbb{R}^{n/2}$$

for all $\alpha \in K^\times$, where $\sigma_1, \overline{\sigma_1}, \dots, \sigma_s, \overline{\sigma_s} : K \rightarrow \mathbb{C}$ are the different embeddings of K into \mathbb{C} . The following is known as *Dirichlet’s unit theorem* [22, Theorem (7.3)].

Theorem 2.1 (Dirichlet’s Unit Theorem). *Let K be an algebraic number field of degree $n = [K : \mathbb{Q}]$ with signature (r, s) . The group $\Gamma := \text{Log}(\mathcal{O}_K^\times)$ is a lattice of dimension $k := r + s - 1$, orthogonal to the vector $\mathbf{1} := (1, \dots, 1) \in \mathbb{R}^{r+2s}$. We call Γ the **log-unit lattice**.*

Lemma 2.2 ([22, (7.1) Proposition]). *For an algebraic number field K it holds that $\ker(\text{Log}|_{\mathcal{O}_K^\times}) = \mu(K)$.*

Theorem 2.1 and Lemma 2.2 imply the following corollary.

Corollary 2.3. *Let K be an algebraic number field of degree $n = [K : \mathbb{Q}]$ with signature (r, s) . The group of units \mathcal{O}_K^\times is isomorphic to $\mu(K) \times \mathbb{Z}^{r+s-1}$, i.e., there are units $\eta_1, \dots, \eta_k \in \mathcal{O}_K^\times$ (where $k := r + s - 1$), such that each $\alpha \in \mathcal{O}_K^\times$ can be written as $\alpha = \zeta \eta_1^{e_1} \cdots \eta_k^{e_k}$ with unique $e_1, \dots, e_k \in \mathbb{Z}$ and $\zeta \in \mu(K)$.*

Such sets $\{\eta_1, \dots, \eta_k\} \subseteq \mathcal{O}_K^\times$ of multiplicative independent units which generates \mathcal{O}_K^\times up to roots of unity like in Corollary 2.3 are called **fundamental systems of units** of \mathcal{O}_K . We now define a “short generator” of a principal fractional ideal.

Definition 2.4. *Let K be an algebraic number field and $g \in K^\times$. Then $g' \in K^\times$ is called a **shortest generator** of the principal fractional ideal $g\mathcal{O}_K$ if $g'\mathcal{O}_K = g\mathcal{O}_K$ and*

$$\| \text{Log}(g') \|_2 = \min_{u \in \mathcal{O}_K^\times} \| \text{Log}(g \cdot u) \|_2 = \min_{u \in \mathcal{O}_K^\times} \| \text{Log}(g) + \text{Log}(u) \|_2.$$

This means g' is a generator of $g\mathcal{O}_K$ with minimal norm in the logarithmic embedding.

2.4 Cyclotomic Fields

A **cyclotomic field** K_m is an algebraic number field of the form $K_m = \mathbb{Q}(\xi_m)$ for some **primitive** m -th root of unity $\xi_m \in \mathbb{C}$, i.e., $\text{ord}(\xi_m) = m$. If $m \not\equiv 2 \pmod{4}$, the number m is called the **conductor** of K_m . The field extension K_m/\mathbb{Q} is Galois with index $[K_m : \mathbb{Q}] = \varphi(m)$, where $\varphi(\cdot)$ is the Euler totient function. The automorphisms $\sigma_i(\cdot)$ of K_m are characterized by $\sigma_i(\xi_m) := \xi_m^i$ for $i \in \mathbb{Z}_m^\times$. Therefore, the Galois group $\text{Gal}(K_m/\mathbb{Q})$ is isomorphic to \mathbb{Z}_m^\times . From now on we fix $\xi_m := e^{2\pi i/m}$ and $K_m := \mathbb{Q}(\xi_m)$ and define $\mathcal{O}_m := \mathcal{O}_{K_m}$. If $m = 2 \cdot k$ for some odd $k \in \mathbb{N}$, we have $\xi_m = -\xi_k$ and therefore $\mathbb{Q}(\xi_m) = \mathbb{Q}(\xi_k)$. Hence, without loss of generality it is sufficient to assume $m \not\equiv 2 \pmod{4}$. The ring of integers is given by $\mathcal{O}_m = \mathbb{Z}[\xi_m]$ (e.g. [22, Proposition (10.2)]).

Lemma 2.5. *For a cyclotomic field K_m we have $\mu(K_m) = \langle \pm \xi_m \rangle = \{ \pm \xi_m^i \mid i \in \mathbb{Z} \}$.*

A proof of the previous lemma can be found in the extended version of this paper [13]. The m -th cyclotomic polynomial $\Phi_m(X) \in \mathbb{Z}[X]$ is defined as the minimal polynomial of the m -th root of unity $\xi_m \in \mathbb{C}$ over \mathbb{Q} . It is given by $\Phi_m(X) = \prod_{i \in \mathbb{Z}_m^\times} (X - \xi_m^i)$. We need the value of the cyclotomic polynomials in $X = 1$.

Lemma 2.6. *Let $m \in \mathbb{N}$ with $m \geq 2$. Then the following holds.*

$$\Phi_m(1) = \begin{cases} p, & \text{if } m = p^l \text{ for some prime } p \text{ and } l \in \mathbb{N} \\ 1, & \text{else.} \end{cases}$$

This lemma is a direct consequence of [12, Corollary 4].

2.5 Circulant Matrices and Characters

We follow along [8, Sect. 2.2] and present some facts about circulant matrices and characters of finite abelian groups.

Definition 2.7 (Circulant matrices). *Let G be a finite abelian group and $\mathbf{a} = (a_g)_{g \in G} \in \mathbb{C}^G$ a complex vector indexed by G . The G -circulant matrix associated with \mathbf{a} is the $G \times G$ matrix*

$$\mathbf{A} := (a_{i \cdot j^{-1}})_{(i,j) \in G \times G} \in \mathbb{C}^{G \times G}.$$

Notice that the transposed matrix of a G -circulant matrix \mathbf{A} associated to $\mathbf{a} = (a_g)_{g \in G}$ is again a G -circulant matrix associated to $\mathbf{a}' = (a_{g^{-1}})_{g \in G}$.

Definition 2.8 (Characters). *Let G be a finite abelian group. A character of G is a group homomorphism $\chi : G \rightarrow \mathbb{S}^1 := \{z \in \mathbb{C} \mid |z| = 1\}$, i.e., $\chi(g \cdot h) = \chi(g) \cdot \chi(h)$ for all $g, h \in G$. The set of all characters of G is denoted by \widehat{G} and forms a group with the usual multiplication of functions, i.e., $(\chi \cdot \Psi)(g) := \chi(g) \cdot \Psi(g)$ for all $\chi, \Psi \in \widehat{G}$ and $g \in G$. The inverse of a character $\chi \in \widehat{G}$ as a group element is given by $\bar{\chi}$, the composition of the complex conjugation and χ . The constant character $\chi \equiv 1$ is the identity element of \widehat{G} and is called **trivial character**. Each finite abelian group G is isomorphic to $\widehat{\widehat{G}}$. In particular, $|G| = |\widehat{G}|$, see [27, Lemma 3.1].*

Theorem 2.9. *Let G be a cyclic group of order n with generator $g \in G$. Then all characters of G are given by $\chi_h(b) := \xi_n^{h \cdot \log_g(b)}$ for $0 \leq h \leq n - 1$, where $\xi_n \in \mathbb{C}$ is a primitive root of unity of order n and $\log_g(b) \in \mathbb{Z}$ with $g^{\log_g(b)} = b \in G$.*

Proof. Let $\chi \in \widehat{G}$ be a character, then $1 = \chi(1) = \chi(g^n) = \chi(g)^n$ holds. Therefore $\chi(g)$ has to be an n -th root of unity. It is easy to see that the functions χ_h are well defined and n different characters. Since there are only $|\widehat{G}| = |G| = n$ different characters, that are all characters of G . □

A **Dirichlet character** $\chi \pmod n$ is a character of the group $G = \mathbb{Z}_n^\times$, for some $n \in \mathbb{N}$. If $n|m$, the character χ of \mathbb{Z}_n^\times induces a character of \mathbb{Z}_m^\times via concatenation of the natural projection $\pi : \mathbb{Z}_m \rightarrow \mathbb{Z}_n$ and χ , i.e., $\chi \circ \pi$. The **conductor** of a character $\chi \in \widehat{\mathbb{Z}_n^\times}$ is defined as the smallest number $f_\chi \in \mathbb{N}$ with $f_\chi | n$, such that χ is induced by some character $\Psi \in \widehat{\mathbb{Z}_{f_\chi}^\times}$. If $n = f_\chi$ for some character $\chi \pmod n$, then χ is called **primitive character**. A character $\chi \in \widehat{\mathbb{Z}_n^\times}$ is said to be **even** if $\chi(-1) = 1$, else we say χ is **odd**. A non-trivial character χ with $\text{Im}(\chi) \in \{\pm 1\}$ is called **quadratic**. We extend a Dirichlet character $\chi : \mathbb{Z}_n^\times \rightarrow \mathbb{S}^1$ of conductor f_χ to a multiplicative function $\chi' : \mathbb{Z} \rightarrow \mathbb{S}^1 \cup \{0\}$ by $\chi'(z) := \chi_{f_\chi}(z)$ if $\text{gcd}(z, f_\chi) = 1$ and zero else, where $\chi_{f_\chi} : \mathbb{Z}_{f_\chi}^\times \rightarrow \mathbb{S}^1$ is a primitive character which induces χ . We just write χ instead of χ' , when needed. We identify characters χ of an arbitrary finite abelian group G with the complex

vector $(\chi(g))_{g \in G} \in \mathbb{C}^G$. This allows for geometrical calculations on characters and provides coherence between circular matrices and characters. For a proof of the following lemma see [8, Sect. 2.2] and use the fact, that $G \cong \widehat{G}$ holds for all finite abelian groups G .

Lemma 2.10. *Let G be a finite abelian group. Then the following holds.*

- (1) For all $\chi \in \widehat{G}$ we have $\sum_{g \in G} \chi(g) = |G|$ if $\chi \equiv 1$ and 0 else.
- (2) All characters $\chi \in \widehat{G}$ have Euclidean norm $\|\chi\|_2 = \sqrt{\langle \chi, \chi \rangle} = \sqrt{|G|}$.
- (3) Different characters $\chi, \Psi \in \widehat{G}$ are pairwise orthogonal, i.e. $\langle \chi, \Psi \rangle = 0$.
- (4) For all $g \in G$ we have $\sum_{\chi \in \widehat{G}} \chi(g) = |G|$ if g is the identity element of G and 0 else.

Definition 2.11. *The **circulant matrix** of a finite abelian group G is defined as*

$$P_G := |G|^{-1/2} \cdot (\chi(g))_{(g,\chi) \in G \times \widehat{G}} \in \mathbb{C}^{G \times \widehat{G}}.$$

It follows directly from Lemma 2.10 that P_G is unitary, i.e., $P_G^{-1} = \overline{P_G}^T$.

Lemma 2.12 ([8, Lemma 2.4]). *Let G be a finite abelian group and $A \in \mathbb{C}^{G \times G}$ be a complex $G \times G$ matrix. The matrix A is G -circulant if and only if the $\widehat{G} \times \widehat{G}$ matrix $P_G^{-1} \cdot A \cdot P_G$ is diagonal; equivalently the columns of P_G are the eigenvectors of A . If A is the G -circulant matrix associated with $\mathbf{a} = (a_g)_{g \in G}$, its eigenvalues corresponding to $\chi \in \widehat{G}$ is $\lambda_\chi = \langle \mathbf{a}, \chi \rangle = \sum_{g \in G} a_g \cdot \overline{\chi(g)}$.*

The following statement is a direct consequence of the previous lemma.

Theorem 2.13. *Let G be a finite abelian group, $\mathbf{a} = (a_g)_{g \in G} \in \mathbb{C}^G$ be a complex vector with associated G -circulant matrix A . The norm of the vector \mathbf{a} is given by*

$$\|\mathbf{a}\|_2^2 = |G|^{-1} \cdot \sum_{\chi \in \widehat{G}} |\lambda_\chi|^2,$$

where $\lambda_\chi = \langle \mathbf{a}, \chi \rangle = \sum_{g \in G} a_g \cdot \overline{\chi(g)}$ is the eigenvalue of A corresponding to the eigenvector χ .

Proof. Since P_G and therefore $\overline{P_G}^T$ is unitary, which means that it is norm preserving, we have

$$\|\mathbf{a}\|_2^2 = \|\overline{P_G}^T \cdot \mathbf{a}\|_2^2 = \sum_{\chi \in \widehat{G}} \left| \sum_{g \in G} a_g \cdot |G|^{-1/2} \overline{\chi(g)} \right|^2 = |G|^{-1} \sum_{\chi \in \widehat{G}} |\lambda_\chi|^2. \quad \square$$

2.6 Dirichlet L-Series

Definition 2.14. Let χ be any Dirichlet character, then the Dirichlet L-function $L(\cdot, \chi)$ is defined as

$$L(\cdot, \chi) : \mathbb{H} \rightarrow \mathbb{C}, \quad s \mapsto L(s, \chi) := \sum_{n \in \mathbb{N}} \frac{\chi(n)}{n^s},$$

where $\mathbb{H} := \{s \in \mathbb{C} \mid \Re(s) > 1\}$.

Since the sum in the definition is absolutely convergent for every $s \in \mathbb{H}$, the sum converges uniformly on every $\mathbb{H}_t := \{s \in \mathbb{C} \mid \Re(s) > t\}$ with $t > 1$. Hence, $L(\cdot, \chi)$ is an analytic function on \mathbb{H} . If χ is the trivial character mod 1, i.e., $\chi(n) = 1$ for all $n \in \mathbb{Z}$, the Dirichlet L-function $L(\cdot, \chi)$ is given by the Riemann zeta function $\zeta(s) = \sum_{n \in \mathbb{N}} \frac{1}{n^s}$. If χ is a non-trivial character mod $m \in \mathbb{N}$, the Dirichlet L-function $L(\cdot, \chi)$ can be extended uniquely to the whole complex plane, see for example [21, Theorem 10.7. ff]. Therefore, $L(1, \chi)$ is well defined in this case.

Theorem 2.15. There exists a constant $C > 0$, such that for every non quadratic Dirichlet character χ mod $m \in \mathbb{N}$ of conductor $f_\chi > 1$

$$|L(1, \chi)| \geq \frac{1}{C \log(f_\chi)},$$

and for every quadratic character χ mod $m \in \mathbb{N}$ of conductor $f_\chi > 1$

$$|L(1, \chi)| \geq \frac{1}{C \sqrt{f_\chi}}.$$

In particular, $L(1, \chi) \neq 0$ if χ is a non-trivial Dirichlet character.

The first inequality was proven by Landau, see [15, p. 29]. For the second inequality, see [25] or [14] for concrete results on the constant $C > 0$.

3 General Algorithmic Approach

In this section we sketch the algorithmic approach and sufficient success conditions presented in [2, 6, 8] to find a shortest generator of some principal fractional ideal, given an arbitrary generator.

A standard approach for recovering a short generator of a principal fractional ideal is shifting this problem to a closest vector problem with requirements to the distance of the target point to the lattice, called **bounded-distance decoding (BDD)**.

Problem 3.1 (BDD). Given a lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ and a target point $\mathbf{t} \in \text{span}(\mathbf{B})$ with the property $\min_{\mathbf{v} \in \mathcal{L}} \|\mathbf{v} - \mathbf{t}\|_2 \leq r$ for some $r < \frac{1}{2} \lambda_1(\mathcal{L})$, where $\lambda_1(\mathcal{L}) := \min_{\mathbf{v} \in \mathcal{L} \setminus \{0\}} \|\mathbf{v}\|_2$, find the unique vector $\mathbf{v} \in \mathcal{L}$ with $\|\mathbf{v} - \mathbf{t}\|_2 \leq r$.

We will use the following **Round-off Algorithm** [1] for solving this problem in our setting.

Algorithm 1. Round-off Algorithm

- 1 **Input:** \mathbf{B}, \mathbf{t} .
 - 2 **Output:** A lattice vector $\mathbf{v} \in \mathcal{L}$.
 - 3 $\mathbf{a} \leftarrow \lfloor (\mathbf{B}^*)^T \cdot \mathbf{t} \rfloor$
 - 4 $\mathbf{v} \leftarrow \mathbf{B} \cdot \mathbf{a}$
 - 5 **return** (\mathbf{v}, \mathbf{a})
-

Lemma 3.2 (Correctness Round-off Algorithm, [8, Claim 2.1]). *Let $\mathcal{L}(\mathbf{B}) \subseteq \mathbb{R}^n$ be a lattice and $\mathbf{t} := \mathbf{v} + \mathbf{e} \in \mathbb{R}^n$ for some $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ and $\mathbf{e} \in \mathbb{R}^n$. If $\langle \mathbf{e}, \mathbf{b}_j^* \rangle \in [-\frac{1}{2}, \frac{1}{2})$ holds for all $j \in \{1, \dots, k\}$, the Round-off Algorithm 1 outputs $\mathbf{v} = \mathbf{B} \cdot \mathbf{a}$ on input \mathbf{B}, \mathbf{t} .*

Note that in general the condition $\langle \mathbf{b}_j^*, \mathbf{e} \rangle \in [-\frac{1}{2}, \frac{1}{2})$ for all $j \in \{1, \dots, k\}$ does not guarantee that the vector \mathbf{v} is in fact the closest vector in $\mathcal{L}(\mathbf{B})$ to $\mathbf{t} = \mathbf{v} + \mathbf{e}$. Therefore, one needs a “sufficiently good” basis \mathbf{B} of the lattice.

Provided that the input basis is sufficiently well suited, Algorithm 2 recovers a shortest generator of a principal fractional ideal in some algebraic number field K .

Algorithm 2. Recovering a short generator with given basis of \mathcal{O}_K^\times

- 1 **Input:** A generator $g' \in K^\times$ of some principal fractional ideal \mathfrak{a} and $b_1, \dots, b_k \in \mathcal{O}_K^\times$ such that $\mathbf{B} := \{\text{Log}(b_1), \dots, \text{Log}(b_k)\}$ is a basis of $\Gamma = \text{Log}(\mathcal{O}_K^\times)$.
 - 2 **Output:** A generator $g_e \in K$ of \mathfrak{a} .
 - 3 $(a_1, \dots, a_k)^T \leftarrow \lfloor (\mathbf{B}^*)^T \cdot \text{Log}(g') \rfloor$ (Round-off-Step)
 - 4 $u' \leftarrow \prod_{i=1}^k b_i^{a_i}$
 - 5 $g_e \leftarrow g'/u'$
 - 6 **return** g_e
-

Lemma 3.3 (Correctness of Algorithm 2, [8, Theorem 4.1]). *Let \mathfrak{a} be a principal fractional ideal in some algebraic number field K of degree $n = [K : \mathbb{Q}]$ with signature (r, s) and $k := r + s - 1$ and let $b_1, \dots, b_k \in \mathcal{O}_K^\times$ be a fundamental system of units of \mathcal{O}_K^\times . Assume that there exists some generator $g \in K^\times$ of \mathfrak{a} satisfying*

$$|\langle \text{Log}(g), \text{Log}(b_i)^* \rangle| < \frac{1}{2} \quad \text{for all } i \in \{1, \dots, k\}.$$

Then for any input generator $g' \in K^\times$ of \mathfrak{a} Algorithm 2 outputs a generator g_e of \mathfrak{a} with same norm as g , i.e., $\|\text{Log}(g)\|_2 = \|\text{Log}(g_e)\|_2$.

Theorem 3.4. *Algorithm 2 has (classical) polynomial running time in $n = [K : \mathbb{Q}]$.*

Proof. Since $k = r + s - 1 \leq n$, the algorithm only computes the $n \times k$ matrix \mathbf{B} , the dual basis \mathbf{B}^* , which includes computing the inverse of a $k \times k$ matrix, and some matrix and vector multiplications of matrices and vectors of size k , which is all polynomial in n . □

One natural question arises: If we draw a generator $g \in K^\times$ from a distribution D over K (without loss of generality we ignore the case $g = 0$), does the condition $|\langle \text{Log}(g), \text{Log}(b_i)^* \rangle| < \frac{1}{2}$ hold for all $i \in \{1, \dots, k\}$ with non-negligible probability $\omega > 0$ for a fixed basis b_1, \dots, b_k ? Lemma 3.3 gives rise to the following definition.

Condition 3.5. *Let D be a probability distribution over some algebraic number field K and $M > 0$. If the probability that for all vectors $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{R}^n$ of Euclidean norm 1 orthogonal to the all-one vector $\mathbf{1} \in \mathbb{R}^n$ the inequalities*

$$|\langle \text{Log}(g), \mathbf{v}_i \rangle| < \frac{1}{2M} \quad \text{for all } i \in \{1, \dots, k\}$$

are satisfied is at least $\omega \in (0, 1)$, where $g \in K$ is drawn from D , we say D satisfies Condition 3.5 with parameters M and ω .

Condition 3.5 can be seen as a sufficient success condition on Algorithm 2, as shown in the following theorem.

Theorem 3.6. *If D is a distribution over an algebraic number field K satisfying Condition 3.5 with parameters $M = \max\{\|\text{Log}(b_1)^*\|_2, \dots, \|\text{Log}(b_k)^*\|_2\}$ and $\omega \in (0, 1)$ for the input basis $b_1, \dots, b_k \in \mathcal{O}_K^\times$ and $g \in K$ is chosen from D , then for any input generator g' of $\mathfrak{a} = g\mathcal{O}_K$, Algorithm 2 outputs a generator $g_e \in K$ of \mathfrak{a} with Euclidean norm at most the norm of g with probability at least $\omega > 0$.*

Proof. We set $v_i := \text{Log}(b_i)^* / \|\text{Log}(b_i)^*\|_2$, which have norm 1 and are orthogonal to the all-one vector $\mathbf{1} \in \mathbb{R}^n$, where $n = [K : \mathbb{Q}]$. Since the distribution D satisfies Condition 3.5 with parameters M and $\omega > 0$ for $b_1, \dots, b_k \in \mathcal{O}_K^\times$, we conclude that

$$|\langle \text{Log}(g), \text{Log}(b_i)^* \rangle| = \|\text{Log}(b_i)^*\|_2 \cdot |\langle \text{Log}(g), \mathbf{v}_i \rangle| < M \frac{1}{2M} = \frac{1}{2}$$

holds with probability ω . □

As shown in [8, Sect. 5] for arbitrary cyclotomic fields $\mathbb{Q}(\xi_m)$ a natural distribution satisfying Condition 3.5 with a not too small parameter $\omega > 0$ for the basis discussed in Sect. 4.2 is the continuous Gaussian distribution. This is a consequence of the following theorem (for more details see [8, 5 Tail Bounds]).

Lemma 3.7 ([8, Lemma 5.4]). *Let $n \in \mathbb{N}$, $X_1, \dots, X_n, X'_1, \dots, X'_n$ be i.i.d. $N(0, \sigma^2)$ variables for some $\sigma > 0$, and let $\widehat{X}_i = (X_i^2 + (X'_i)^2)^{1/2}$ for $i \in \{1, \dots, n\}$. Then for any set of $l \in \mathbb{N}$ vectors $\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(l)} \in \mathbb{R}^n$ of Euclidean norm 1 that are orthogonal to the all-one vector $\mathbf{1} \in \mathbb{R}^n$ and every $t \geq C_\sigma$ for some universal constant C_σ (that only depends on σ) it holds that*

$$\Pr \left[\exists j : \left| \left\langle \left(\log(\widehat{X}_1), \dots, \log(\widehat{X}_n) \right)^T, \mathbf{a}^{(j)} \right\rangle \right| \geq t \right] \leq 2l \exp \left(-\frac{t}{4} \right).$$

Applied to our setting of cyclotomic number fields, we obtain that Condition 3.5 is satisfied for Gaussian distributions if the norms of the basis elements in the logarithmic embedding are sufficiently short.

Corollary 3.8. *Let $m \in \mathbb{N}$, $m \geq 3$, $n = \varphi(m)$, and $k = n/2 - 1$. If $M := \max\{\| \text{Log}(b_j)^* \|_2, \dots, \| \text{Log}(b_k)^* \|_2\}$ is small enough, i.e., $1/2M \geq C_\sigma$, Condition 3.5 is satisfied for Gaussian distributions (with standard deviation σ) with parameters M and $\omega(m) = 1 - 2k \exp(-\frac{1}{8M})$, if $\omega(m) > 0$.*

There is one issue with this approach. Algorithm 2 uses a basis b_1, \dots, b_k of \mathcal{O}_K^\times (up to roots of unity), i.e., a fundamental set of units, with sufficiently short dual vectors. However, in general, given a number field K , such basis is not known. Instead, for special instances of cyclotomic number fields $K = \mathbb{Q}(\xi_m)$, namely if m is a prime-power or a product of two prime powers (as analyzed in the next section), only a well suited basis $b_1, \dots, b_k \in \mathcal{O}_m^\times$ of a subgroup F with finite index in \mathcal{O}_m^\times is known. This can be compensated for by computing a fundamental system of units of \mathcal{O}_K^\times and afterwards a set of representatives $u_1, \dots, u_f \in \mathcal{O}_K^\times$ of $\mathcal{O}_K^\times / \mu(K)F$, using classical [3] or quantum [10] algorithms. The quantum algorithm has running time polynomial in $n = [K : \mathbb{Q}]$ and $\log(|d_K|)$, where d_K denotes the discriminant of K . Notice, if $K = \mathbb{Q}(\xi_m)$ is a cyclotomic field, we obtain $|d_K| \in O(n \log(m))$ as a direct consequence of [27, Proposition 2.7]. Hence, the quantum algorithm runs in polynomial time in m . Note that the calculation of the set of representatives $u_1, \dots, u_f \in \mathcal{O}_K^\times$ of $\mathcal{O}_K^\times / \mu(K)F$ has to be done only once for each cyclotomic field $K = \mathbb{Q}(\xi_m)$ and can therefore be seen as precomputation cost. If one has computed such a set of representatives $u_1, \dots, u_f \in \mathcal{O}_K^\times$, we can enumerate over all of them and apply Algorithm 2 for each g'/u_i , increasing the running time only by the factor $f := |\mathcal{O}_K^\times / \mu(K)F|$. The detailed algorithm if one has precomputed such a set of representatives can be found in the extended version of this paper [13].

In this work, we show that for cyclotomic number fields $\mathbb{Q}(\xi_m)$ the index of the basis presented in Sect. 4.2 is polynomial in m , if $m = p^\alpha q^\beta$ for some suitable odd primes p and q . This yields a polynomial running time in m of Algorithm 2 in this case.

4 Finding Shortest Generators in Cyclotomic Fields of Conductor $m = p^\alpha q^\beta$

In this section we study the SG-PIP in cyclotomic fields of composite conductor $m = p^\alpha q^\beta$ for distinct odd primes p, q .

4.1 Generator Prime Pairs

In the next section we investigate the group generated by the elements $\frac{\xi_m^u - 1}{\xi_m - 1} \in \mathcal{O}_m^\times$ with $j \in \mathbb{Z}_m^\times$ in the case where $m = p^\alpha q^\beta$ has only two distinct odd prime factors. We show that the index of this group in the full group of units is finite iff

p is a generator of $\mathbb{Z}_{q^\beta}^\times$ or a square of a generator and q is a generator of $\mathbb{Z}_{p^\alpha}^\times$ or a square of a generator. Therefore, we introduce the following notion and derive several results surrounding it.

Definition 4.1. *Let $\alpha, \beta \in \mathbb{N}$ and $p, q \in \mathbb{P}$ be two distinct odd primes with the following properties:*

- (i) • *If $q - 1 \equiv 0 \pmod{4}$: p is a generator of $\mathbb{Z}_{q^\beta}^\times$.*
- *If $q - 1 \not\equiv 0 \pmod{4}$: p is a generator of $\mathbb{Z}_{q^\beta}^\times$ or has order $\frac{\varphi(q^\beta)}{2} = q^{\beta-1} \cdot \frac{q-1}{2}$ in $\mathbb{Z}_{q^\beta}^\times$.*

And

- (ii) • *If $p - 1 \equiv 0 \pmod{4}$: q is a generator of $\mathbb{Z}_{p^\alpha}^\times$.*
- *If $p - 1 \not\equiv 0 \pmod{4}$: q is a generator of $\mathbb{Z}_{p^\alpha}^\times$ or has order $\frac{\varphi(p^\alpha)}{2} = p^{\alpha-1} \cdot \frac{p-1}{2}$ in $\mathbb{Z}_{p^\alpha}^\times$.*

We call such a pair (p, q) an (α, β) -**generator prime pair** ((α, β) -**GPP**). If (p, q) is an (α, β) -generator prime pair for every $\alpha, \beta \in \mathbb{N}$, we just say that (p, q) is a **generator prime pair** (**GPP**).

The definition of generator prime pairs is useless for testing given prime pairs on this property, since infinitely many pairs of α and β have to be checked. To obtain a better criterion, we use the following result.

Theorem 4.2 ([7, Lemma 1.4.5]). *Let p be an odd prime, and let $g \in \mathbb{Z}$ be a primitive root modulo p . Then either g or $g + p$ is a primitive root modulo every power of p .*

In particular, if $g \in \mathbb{Z}$ is a generator of $\mathbb{Z}_{p^2}^\times$ and therefore also for \mathbb{Z}_p^\times , then g is a generator for all $\mathbb{Z}_{p^l}^\times$ with $l \in \mathbb{N}$.

A direct consequence of Theorem 4.2 is that $\mathbb{Z}_{p^l}^\times$ is cyclic for every $l \in \mathbb{N}$ and odd prime number $p \in \mathbb{P}$, which implies the following corollaries. The proofs can be found in the extended version of this paper [13].

Corollary 4.3. *Let p be an odd prime, $l \in \mathbb{N}$ and $g \in \mathbb{Z}_{p^l}^\times$ be a generator. Then the even Dirichlet characters of $\mathbb{Z}_{p^l}^\times$ are given by $\chi_h(b) := \xi_{\varphi(p^l)}^{h \cdot a(b)}$ for $0 \leq h \leq \varphi(p^l) - 1$ and h is even, where $\xi_{\varphi(p^l)} \in \mathbb{C}$ is a primitive root of unity of order $\varphi(p^l)$ and $a(b) \in \mathbb{Z}$ with $g^{a(b)} = b \in \mathbb{Z}_{p^l}^\times$.*

Corollary 4.4. *Let (p, q) be an (α, β) -GPP for some $\alpha, \beta \in \mathbb{N}$ and $\beta \geq 2$. Then (p, q) is an (α, l) -GPP for every $l \in \mathbb{N}$. Analogously, the same results follows if we swap p and q .*

In particular, (p, q) is a GPP iff it is a $(2, 2)$ -GPP.

With Corollary 4.4 we can easily check prime pairs if they are generator prime pairs by testing if they are a $(2, 2)$ -GPP.

In the extended version of this paper [13] we provide some examples and numerical data of generator prime pairs. By computation, more than 35% of all odd prime pairs up to 32600 are generator prime pairs, see Fig. 1. An interesting fact is that a similar notion of prime pairs was used in the proof of Catalan’s conjecture by Mihailescu [19], namely **double Wieferich prime pairs** (p, q) , which satisfy

$$p^{q-1} \equiv 1 \pmod{q^2} \text{ and } q^{p-1} \equiv 1 \pmod{p^2},$$

see [24, Chap. 1]. More information about their relation can be found in the extended version of this paper [13].

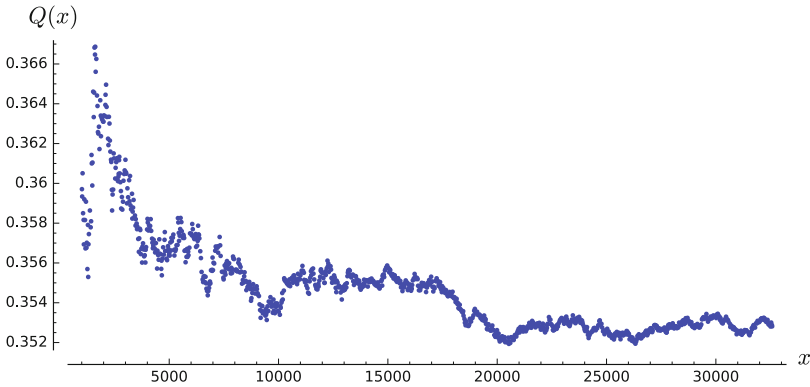


Fig. 1. Values of the quotient $Q(x) = \frac{\text{Number of GPP } (p,q) \text{ with } 2 < p < q \leq x}{\text{Number of prime pairs } (p,q) \text{ with } 2 < p < q \leq x}$

4.2 Suitable Units in the Case $m = p^\alpha q^\beta$

Let $m \in \mathbb{N}$ with $m \geq 3$. For the rest of this section, for $j \in \mathbb{Z}_m^\times$ let

$$b_j := \frac{\xi_m^j - 1}{\xi_m - 1} \in \mathcal{O}_m^\times \text{ and } \mathbf{b}_j := \text{Log}_r(b_j) \in \mathbb{R}^{n/2}, \tag{1}$$

where $n = \varphi(m)$. Further, let $G_m := \mathbb{Z}_m^\times / \{\pm 1\}$ (one can identify the group G_m with the set of representatives $\{l \in \mathbb{N} \mid 1 \leq l < \frac{m}{2} \text{ with } \gcd(l, m) = 1\}$) and let \mathcal{S}_m denote the group generated by $\{b_j \mid j \in G_m \setminus \{1\}\}$ and $\pm \xi_m$, i.e., we collect the vectors \mathbf{b}_j for $j \in G_m \setminus \{1\}$ in the matrix

$$\mathbf{B} := \left(\log \left(\left| \frac{\xi_m^{ij} - 1}{\xi_m^i - 1} \right| \right) \right)_{\substack{i \in G_m \\ j \in G_m \setminus \{1\}}} . \tag{2}$$

Notice that $b_{-j} = \xi_m^a \cdot b_j$ for some $a \in \mathbb{Z}_m$, hence it is sufficient to consider a set of representatives of $\{b_j \mid j \in G_m \setminus \{1\}\}$ as generators of \mathcal{S}_m . The characters of

$G_m = \mathbb{Z}_m^\times / \{\pm 1\}$ correspond to the even characters of \mathbb{Z}_m^\times via concatenation with the canonical projection $\mathbb{Z}_m^\times \rightarrow \mathbb{Z}_m^\times / \{\pm 1\}$. We identify the characters of G_m with the even characters of \mathbb{Z}_m^\times .

If $[\mathcal{O}_m^\times : \mathcal{S}_m]$ is finite, the elements b_j for $j \in G_m \setminus \{1\}$ have to be a basis of the group \mathcal{S}_m , by comparing the \mathbb{Z} -rank of \mathcal{S}_m and \mathcal{O}_m^\times , which is $\frac{\varphi(m)}{2} - 1 = |G_m \setminus \{1\}|$.

4.3 Index of the Subgroup in the Full Unit Group

We determine the index of \mathcal{S}_m in the full group of units \mathcal{O}_m^\times in the case $m = p^\alpha q^\beta$ with $\alpha, \beta \in \mathbb{N}$ and distinct odd primes p, q . As we show in this work, the index is finite iff (p, q) is an (α, β) -generator prime pair. Moreover, in this case the index is bounded by the product of the class number h_m^+ and a factor, which is linear in m .

The next lemma provides an explicit expression for the index of \mathcal{S}_m in the full group of units \mathcal{O}_m^\times , which is a direct consequence of [27, Corollary 8.8].

Lemma 4.5. *Let $m \in \mathbb{N}$ with $m \geq 3$ and $m \not\equiv 2 \pmod{4}$. If m is not a prime-power, i.e., has at least two distinct prime factors, the index of \mathcal{S}_m in \mathcal{O}_m^\times is given by*

$$[\mathcal{O}_m^\times : \mathcal{S}_m] = 2h_m^+ \prod_{\substack{\chi \in \widehat{G}_m \\ \chi \neq 1}} \prod_{\substack{p|m \\ p \in \mathbb{P}}} (1 - \chi(p))$$

if the right hand side is not equal 0, else the index is infinite. The factor h_m^+ is the class number of $\mathbb{Q}(\xi_m)^+ := \mathbb{Q}(\xi_m + \xi_m^{-1})$.

For $m \in \mathbb{N}$ we define

$$\beta_m := \prod_{\substack{\chi \in \widehat{G}_m \\ \chi \neq 1}} \prod_{\substack{p|m \\ p \in \mathbb{P}}} (1 - \chi(p)).$$

Theorem 4.6. *Let p, q be two distinct odd primes and $m = p^\alpha q^\beta$ for some $\alpha, \beta \in \mathbb{N}$. Then*

$$\beta_m = \frac{\varphi(m)}{4} = \frac{(p-1)(q-1)}{4pq} m$$

iff (p, q) is an (α, β) -GPP, and $\beta_m = 0$ otherwise. In particular, the index is finite and bounded by $[\mathcal{O}_m^\times : \mathcal{S}_m] = h_m^+ \frac{(p-1)(q-1)}{2pq} m \leq h_m^+ \cdot \frac{m}{2}$, iff (p, q) is an (α, β) -GPP.

Proof. Assume that (p, q) is an (α, β) -generator prime pair. Since m is only divisible by the primes p, q , we obtain

$$\beta_m = \prod_{\substack{\chi \in \widehat{G}_m \\ \chi \neq 1}} \prod_{\substack{t|m \\ t \in \mathbb{P}}} (1 - \chi(p)) = \left(\prod_{\substack{\chi \in \widehat{G}_{p^\alpha} \\ \chi \neq 1}} (1 - \chi(q)) \right) \cdot \left(\prod_{\substack{\chi \in \widehat{G}_{q^\beta} \\ \chi \neq 1}} (1 - \chi(p)) \right),$$

because $\chi(p) = \chi(q) = 0$ and therefore $(1 - \chi(p))(1 - \chi(q)) = 1$ if $pq|f_\chi$. Hence it is sufficient to prove

$$\prod_{\substack{\chi \in \widehat{G}_{q^\beta} \\ \chi \neq 1}} (1 - \chi(p)) = \frac{\varphi(q^\beta)}{2}.$$

Let g be a generator of $\mathbb{Z}_{q^\beta}^\times$, and $a \in \mathbb{Z}$ with $g^a \equiv p \pmod{q^\beta}$. Since (p, q) is an (α, β) -generator prime pair, we conclude $\gcd\left(a, \frac{\varphi(q^\beta)}{2}\right) = 1$ by comparing the order of p in $\mathbb{Z}_{q^\beta}^\times$, independent whether $q - 1 \equiv 0 \pmod{4}$ or $q - 1 \not\equiv 0 \pmod{4}$. The even characters of $\mathbb{Z}_{q^\beta}^\times$ are given by Corollary 4.3, which implies

$$\begin{aligned} \prod_{\substack{\chi \in \widehat{G}_{q^\beta} \\ \chi \neq 1}} (1 - \chi(p)) &= \prod_{\substack{1 \leq h \leq \varphi(q^\beta) - 1 \\ h \text{ even}}} (1 - \xi_{\varphi(q^\beta)}^{ha}) \stackrel{(1)}{=} \prod_{1 \leq k \leq \frac{\varphi(q^\beta)}{2} - 1} \left(1 - \xi_{\frac{\varphi(q^\beta)}{2}}^k\right) \\ &\stackrel{(2)}{=} \frac{X^{\frac{\varphi(q^\beta)}{2}} - 1}{X - 1} \Big|_{X=1} = \left(X^{\frac{\varphi(q^\beta)}{2} - 1} + X^{\frac{\varphi(q^\beta)}{2} - 2} + \dots + 1\right) \Big|_{X=1} = \frac{\varphi(q^\beta)}{2}, \end{aligned}$$

where we used in equality (1) that multiplying with a is a permutation of $\mathbb{Z}_{\frac{\varphi(q^\beta)}{2}}$ with $0 \cdot a \equiv 0 \pmod{\frac{\varphi(q^\beta)}{2}}$, since $\gcd\left(a, \frac{\varphi(q^\beta)}{2}\right) = 1$, and in (2) we used $X^l - 1 = \prod_{0 \leq k \leq l-1} (X - \xi_l^k)$ for all $l \in \mathbb{N}$.

Conversely, assume that (p, q) is not an (α, β) -generator prime pair, i.e., without loss of generality p is not a generator of $\mathbb{Z}_{q^\beta}^\times$ and has not order $\frac{\varphi(q^\beta)}{2}$ in $\mathbb{Z}_{q^\beta}^\times$ if $q - 1 \not\equiv 0 \pmod{4}$. Again, let g be a generator of $\mathbb{Z}_{q^\beta}^\times$, and $a \in \mathbb{Z}$ with $g^a \equiv p \pmod{q^\beta}$. We conclude that $\gcd\left(a, \frac{\varphi(q^\beta)}{2}\right) > 1$ holds. Hence, there exists a prime number $t \in \mathbb{P}$ such that $t | \gcd\left(a, \frac{\varphi(q^\beta)}{2}\right)$ holds. Then $h := \frac{\varphi(q^\beta)}{t} \in \mathbb{N}$ is even and $1 \leq h \leq \varphi(q^\beta) - 1$. By Corollary 4.3, there is a non-trivial, even Dirichlet character χ_h of $\mathbb{Z}_{q^\beta}^\times$ with

$$\chi_h(p) = \xi_{\varphi(q^\beta)}^{ah} = \xi_{\varphi(q^\beta)}^{\frac{a}{t} \varphi(q^\beta)} = 1,$$

which implies $\beta_m = 0$ in this case. □

We have proven that the factor β_m is sufficiently small, if $m = p^\alpha q^\beta$ for some (α, β) -generator prime pair (p, q) . The second factor of the index $[\mathcal{O}_m^\times : \mathcal{S}_m]$ is given by the class number h_m^+ , which has to be sufficiently small, too.

Theorem 4.7 ([20, Theorem 1.1]). *Let m be a composite integer, $m \not\equiv 2 \pmod 4$, and let $\mathbb{Q}(\xi_m)^+$ denote the maximal real subfield of the m -th cyclotomic field $\mathbb{Q}(\xi_m)$. Then the class number h_m^+ of $\mathbb{Q}(\xi_m)^+$ is*

$$h_m^+ = \begin{cases} 1 & \text{if } \varphi(m) \leq 116 \text{ and } m \neq 136, 145, 212, \\ 2 & \text{if } m = 136, \\ 2 & \text{if } m = 145, \\ 1 & \text{if } m = 256, \end{cases}$$

where $\varphi(\cdot)$ is the Euler phi function. Furthermore, under the generalized Riemann hypothesis (GRH), $h_{212}^+ = 5$ and $h_{512}^+ = 1$.

Remark 4.8. *In our case, $m = p^\alpha q^\beta$ for some (α, β) -generator prime pair (p, q) . Since we want a polynomial running time in m of Algorithm 2 for cyclotomic fields $K_m = \mathbb{Q}(\xi_m)$, we need a polynomial bound of the index $[\mathcal{O}_m^\times : \mathcal{S}_m] = 2h_m^+ \beta_m$. The factor $\beta_m \in \mathbb{N}$ is bounded by $\frac{m}{4}$, hence it is sufficient if h_m^+ is bounded by some polynomial in m , if $m = p^\alpha q^\beta$, at least for a fixed generator prime pair (p, q) . We do not know if such a bound holds. However, by Theorem 4.7 one could conjecture that the class number h_m^+ is bounded by some polynomial. In [9] this is presented as a reasonable conjecture.*

4.4 Norms of the Basis Elements

We determine the norm of the dual vectors \mathbf{b}_j^* for $j \in G_m \setminus \{1\}$ in the case, that $m = p^\alpha q^\beta$, for some $\alpha, \beta \in \mathbb{N}$ and (p, q) is an (α, β) -generator prime pair. Again, we follow along [8, Chap. 3].

Let $m \in \mathbb{N}$ with $m \geq 2$. We define

$$z_j := \xi_m^j - 1 \in \mathcal{O}_m \quad \text{and} \quad \mathbf{z}_j := \text{Log}_r(z_j) \in \mathbb{R}^{n/2}$$

for all $j \in \mathbb{Z}_m^\times$ (again, $n = \varphi(m)$). Note that \mathbf{z}_j is well defined since $\xi_m^{-j} - 1$ is the complex conjugate of $\xi_m^j - 1$. We collect all the vectors $\mathbf{z}_{j^{-1}}$ for $j \in G_m$ in the matrix $\mathbf{Z} \in \mathbb{R}^{n/2 \times n/2}$, i.e.,

$$\mathbf{Z} := \left(\log \left(\left| \xi_m^{i \cdot j^{-1}} - 1 \right| \right) \right)_{i, j \in G_m}.$$

Since the entry with index $(i, j) \in G_m \times G_m$ only depends on $i \cdot j^{-1}$, the matrix \mathbf{Z} is G_m -circulant and associated with \mathbf{z}_1 . Notice that the vectors \mathbf{z}_j and the matrix \mathbf{Z} only depend on m .

Our first goal is to prove that only the eigenvalue of \mathbf{Z} corresponding to the trivial character of \mathbb{Z}_m^\times is zero, in the case that $m = p^\alpha q^\beta$, for some $\alpha, \beta \in \mathbb{N}$ and distinct primes p and q .

Lemma 4.9. *Let $m = p^\alpha q^\beta$ for some distinct primes $p, q \in \mathbb{P}$ and $\alpha, \beta \in \mathbb{N}$. Then the eigenvalue λ_χ of \mathbf{Z} corresponding to the trivial character $1 \equiv \chi \in G_m$ is $\lambda_\chi = 0$.*

Proof. By Theorem 2.13, the eigenvalue of the G_m -circulant matrix \mathbf{Z} corresponding to the trivial character $1 \equiv \chi \in G_m$ is given by

$$\lambda_\chi = \langle \mathbf{z}_1, 1 \rangle = \frac{1}{2} \sum_{j \in \mathbb{Z}_m^\times} \log(|\xi_m^j - 1|) = \frac{1}{2} \log \left(\left| \prod_{j \in \mathbb{Z}_m^\times} (\xi_m^j - 1) \right| \right) = \frac{1}{2} \log(|\Phi_m(1)|) \stackrel{(1)}{=} 0,$$

where (1) follows from Lemma 2.6. □

Lemma 4.10. *Let $m = p^\alpha q^\beta$ for some distinct primes $p, q \in \mathbb{P}$ and $\alpha, \beta \in \mathbb{N}$. Furthermore, let $\chi \in \widehat{G}_m$ be an even character of conductor $f_\chi > 1$ with $pq | f_\chi$. Then the eigenvalue λ_χ of \mathbf{Z} corresponding to χ is given by*

$$\lambda_\chi = \frac{1}{2} \sum_{a \in \mathbb{Z}_{f_\chi}^\times} \bar{\chi}(a) \cdot \log(|1 - \xi_{f_\chi}^a|).$$

This can be proven similar to the prime power case in [8, Corollary 3.4], a proof can be found in the extended version of this paper [13].

Lemma 4.11. *Let $m = p^\alpha q^\beta$ for some distinct primes $p, q \in \mathbb{P}$ and $\alpha, \beta \in \mathbb{N}$. Furthermore, let $\chi \in \widehat{G}_m$ be an even character of conductor $f_\chi > 1$ with $q \nmid f_\chi$. Then the eigenvalue λ_χ of \mathbf{Z} corresponding to χ is given by*

$$\lambda_\chi = \frac{1}{2} (1 - \bar{\chi}(q)) \sum_{a \in \mathbb{Z}_{f_\chi}^\times} \bar{\chi}(a) \cdot \log(|1 - \xi_{f_\chi}^a|).$$

Analogously, the same results hold if we swap p and q .

Proof. Let $f := f_\chi > 1$ be the conductor of χ , i.e., $f = p^e$ for some $1 \leq e \leq \alpha$. Further, let $\pi : \mathbb{Z}_m^\times \rightarrow \mathbb{Z}_f^\times$ be the canonical projection. For $a \in \mathbb{Z}_f^\times$ and a fixed integer representative $a' \in \mathbb{Z}$ of $a \in \mathbb{Z}_f^\times$ we have $\pi^{-1}(a) = \Psi^{-1} \left(\left\{ a' + k \cdot f \in \mathbb{Z}_{p^\alpha}^\times \mid 0 \leq k < \frac{p^\alpha}{f} \right\} \times \mathbb{Z}_{q^\beta}^\times \right) \subseteq \mathbb{Z}_m^\times$ by Chinese remainder theorem, where $\Psi : \mathbb{Z}_m \rightarrow \mathbb{Z}_{p^\alpha} \times \mathbb{Z}_{q^\beta}, a \mapsto (a \bmod p^\alpha, a \bmod q^\beta)$. There exists $r_1, r_2 \in \mathbb{Z}$ such that $r_1 q^\beta \equiv 1 \pmod{p^\alpha}$ and $r_2 p^\alpha \equiv 1 \pmod{q^\beta}$, which yields

$$\pi^{-1}(a) = \left\{ (a' + k \cdot f) \cdot r_1 q^\beta + y \cdot r_2 p^\alpha \in \mathbb{Z}_m^\times \mid 0 \leq k < \frac{p^\alpha}{f}, y \in \mathbb{Z}_{q^\beta}^\times \right\} \subseteq \mathbb{Z}_m^\times \tag{3}$$

for a fixed integer representative $a' \in \mathbb{Z}$ of $a \in \mathbb{Z}_f^\times$. We obtain

$$\begin{aligned} \prod_{\substack{j \in \mathbb{Z}_m^\times \\ \pi(b)=a}} (1 - \xi_m^j) &= \prod_{y \in \mathbb{Z}_{q^\beta}^\times} \prod_{0 \leq k < \frac{p^\alpha}{f}} \left(1 - \xi_{p^\alpha}^{kr_1} \cdot \xi_{q^\beta}^{yr_2} \cdot \xi_{p^\alpha}^{a'r_1} \right) \stackrel{(1)}{=} \prod_{y \in \mathbb{Z}_{q^\beta}^\times} \left(1 - \xi_{q^\beta}^{yr_2 \frac{p^\alpha}{f}} \cdot \xi_{p^\alpha}^{a'r_1 \frac{p^\alpha}{f}} \right) \\ &\stackrel{(2)}{=} \prod_{y \in \mathbb{Z}_{q^\beta}^\times} \left(1 - \xi_{q^\beta}^{y \frac{p^\alpha}{f}} \cdot \xi_{p^\alpha}^{ar_1} \right) \stackrel{(3)}{=} \frac{1 - \xi_f^{ar_1 q^\beta}}{1 - \xi_f^{ar_1 q^{\beta-1}}} \stackrel{(4)}{=} \frac{1 - \xi_f^a}{1 - \xi_f^{ar_1 q^{\beta-1}}}. \end{aligned}$$

In equation (1) we have used again the identity $X^n - Y^n = \prod_{0 \leq k < n} (X - \xi_n^k Y)$ for $n := \frac{p^\alpha}{f}$, $X := 1$ and $Y := \xi_{q^\beta}^{yr_2} \cdot \xi_{p^\alpha}^{ar_1}$, where $r_1 \in \mathbb{Z}_{\frac{p^\alpha}{f}}^\times$ and therefore multiplication with r_1 is a permutation of $\mathbb{Z}_{\frac{p^\alpha}{f}}$. The same permutation argument implies equation (2), since $r_2 \in \mathbb{Z}_{q^\beta}^\times$. In (3) we have used the identity $\prod_{a \in \mathbb{Z}_{q^\beta}^\times} (X - \xi_{q^\beta}^a Y) = \frac{X^{q^\beta} - Y^{q^\beta}}{X^{q^{\beta-1}} - Y^{q^{\beta-1}}}$ for $X = 1$ and $Y = \xi_f^{ar_1}$. The hypothesis $r_1 q^\beta \equiv 1 \pmod{p^\alpha}$ implies $r_1 q^\beta \equiv 1 \pmod{f}$ and therefore equation (4).

Finally, we can calculate the eigenvalue λ_χ .

$$\begin{aligned} \lambda_\chi &= \langle \mathbf{z}_1, \chi \rangle = \frac{1}{2} \sum_{j \in \mathbb{Z}_m^\times} \bar{\chi}(j) \cdot \log(|1 - \xi_m^j|) = \frac{1}{2} \sum_{a \in \mathbb{Z}_f^\times} \bar{\chi}(a) \sum_{\substack{j \in \mathbb{Z}_m^\times \\ \pi(j)=a}} \log(|1 - \xi_m^j|) \\ &= \frac{1}{2} \sum_{a \in \mathbb{Z}_f^\times} \bar{\chi}(a) \log \left(\left| \prod_{\substack{j \in \mathbb{Z}_m^\times \\ \pi(j)=a}} (1 - \xi_m^j) \right| \right) = \frac{1}{2} \sum_{a \in \mathbb{Z}_f^\times} \bar{\chi}(a) \log \left(\left| \frac{1 - \xi_f^a}{1 - \xi_f^{ar_1 q^{\beta-1}}} \right| \right) \\ &= \frac{1}{2} \sum_{a \in \mathbb{Z}_f^\times} \bar{\chi}(a) \log(|1 - \xi_f^a|) - \frac{1}{2} \sum_{a \in \mathbb{Z}_f^\times} \bar{\chi}(a) \log(|1 - \xi_f^{ar_1 q^{\beta-1}}|) \\ &\stackrel{(5)}{=} \frac{1}{2} \sum_{a \in \mathbb{Z}_f^\times} \bar{\chi}(a) \log(|1 - \xi_f^a|) - \frac{1}{2} \sum_{a \in \mathbb{Z}_f^\times} \bar{\chi}(a \cdot q) \log(|1 - \xi_f^a|) \\ &= \frac{1}{2} (1 - \bar{\chi}(q)) \sum_{a \in \mathbb{Z}_f^\times} \bar{\chi}(a) \log(|1 - \xi_f^a|), \end{aligned}$$

where we used in (5) the substitution a for $ar_1 q^{\beta-1}$ and the fact, that $r_1 q^\beta \equiv 1 \pmod{p^\alpha}$ implies $r_1 q^{\beta-1} \cdot q \equiv r_1 q^\beta \equiv 1 \pmod{f}$, i.e., q is the multiplicative inverse of $r_1 q^{\beta-1} \pmod{f}$. □

The next theorem provides a connection between the occurring sum in the eigenvalues λ_χ and the Dirichlet L-function.

Theorem 4.12 ([27, Lemma 4.8. and Theorem 4.9]). *Let χ be an even Dirichlet character mod $m \in \mathbb{N}$ of conductor $f_\chi > 1$. Then*

$$\left| \sum_{a \in \mathbb{Z}_{f_\chi}^\times} \bar{\chi}(a) \cdot \log(|1 - \xi_{f_\chi}^a|) \right| = \sqrt{f_\chi} \cdot |L(1, \chi)|.$$

We collect the previous results in the following theorem. A proof can be found in the extended version of this paper [13].

Theorem 4.13. *Let $m = p^\alpha q^\beta$ for some distinct primes $p, q \in \mathbb{P}$ and $\alpha, \beta \in \mathbb{N}$. Further, let $\chi \in \widehat{G}_m$ be an even Dirichlet character mod m of conductor $f_\chi > 1$. Then the eigenvalue $\lambda_\chi = \langle \mathbf{z}_1, \chi \rangle$ of \mathbf{Z} corresponding to χ is given by*

$$|\lambda_\chi| = \frac{1}{2} \left| (1 - \bar{\chi}(p)) (1 - \bar{\chi}(q)) \right| \cdot \sqrt{f_\chi} \cdot |L(1, \chi)|.$$

In particular, if p, q are odd primes, all eigenvalues λ_χ corresponding to some non-trivial even character $\chi \in \widehat{G_m}$ are non-zero iff (p, q) is an (α, β) -generator prime pair.

We are now prepared to express the norm of the dual vectors \mathbf{b}_j^* in terms of the eigenvalues λ_χ . Notice that this is the same result as in the prime-power case, but is more complicated to prove since \mathbf{Z} is not invertible, see [8, Lemma 3.2].

Lemma 4.14. *Let (p, q) be an (α, β) -generator prime pair, and $m := p^\alpha q^\beta$. Then the norm of \mathbf{b}_j^* for all $j \in G_m \setminus \{1\}$ is given by*

$$\|\mathbf{b}_j^*\|_2^2 = |G_m|^{-1} \cdot \sum_{\substack{\chi \in \widehat{G_m} \\ \chi \neq 1}} |\lambda_\chi|^{-2},$$

where $\lambda_\chi = \langle \mathbf{z}_1, \chi \rangle$ denotes the eigenvalue of \mathbf{Z} corresponding to χ . In particular, all dual vectors \mathbf{b}_j^* have the same norm.

Proof. Our goal is to prove the claim by defining a “pseudo inverse” \mathbf{D} of \mathbf{Z}^T and show that \mathbf{b}_j^* is the j -th column of \mathbf{D} .

For simplification, we fix an order of $\widehat{G_m}$, i.e., $\widehat{G_m} = \{\chi_1, \dots, \chi_n\}$ with $n = \frac{\varphi(m)}{2}$ and $\chi_1 \equiv 1$ is the trivial character mod m . This allows us to represent $\widehat{G_m} \times \widehat{G_m}$ matrices by $n \times n$ matrices. Notice that the characters χ_j are different from the characters of Theorem 2.9, we only used a similar notation. The order of $\widehat{G_m}$ yields an order of the eigenvalues $\lambda_1, \dots, \lambda_k$ of \mathbf{Z} , where $\lambda_1 = 0$ by Lemma 4.9 and $\lambda_j \neq 0$ for $2 \leq j \leq n$ by Theorem 4.13. Since \mathbf{Z} is a G_m -circulant matrix, Lemma 2.12 implies

$$\mathbf{Z} = \mathbf{P}_{G_m} \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix} \mathbf{P}_{G_m}^{-1}.$$

We define

$$\mathbf{D}^T := \mathbf{P}_{G_m} \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\lambda_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\lambda_n} \end{pmatrix} \mathbf{P}_{G_m}^{-1} \quad \text{and} \quad \mathbf{Z}_1^M := \mathbf{Z} \begin{pmatrix} 1 & 1 & \dots & 1 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} = (\mathbf{z}_1, \dots, \mathbf{z}_1) \in \mathbb{R}^{G_m \times G_m},$$

where the first row of the matrix, which only has ones in the first row and zeroes elsewhere, corresponds to $1 \in G_m$.

Let \mathbf{d}_j be the j -th column of \mathbf{D} for $j \in G_m$. We claim that $\mathbf{d}_j = \mathbf{b}_j^*$ for all $j \in G_m \setminus \{1\}$. Since $\text{span}(\mathbf{B}) \subseteq \mathbb{R}^{G_m} \cong \mathbb{R}^n$ is the subspace orthogonal to the all-one vector $\mathbf{1}$, we have to prove $\langle \mathbf{d}_j, \mathbf{1} \rangle = 0$ or all $j \in G_m \setminus \{1\}$, first. The

components of the vector \mathbf{d}_j just differ by the order of the entries of \mathbf{d}_1 , since \mathbf{D} is a G_m -circulant matrix associated to \mathbf{d}_1 by Lemma 2.12. Hence,

$$\langle \mathbf{d}_j, \mathbf{1} \rangle = \langle \mathbf{d}_1, \mathbf{1} \rangle = 0,$$

since $\langle \mathbf{d}_1, \mathbf{1} \rangle$ is the eigenvalue of \mathbf{D} corresponding to the trivial character $1 \equiv \chi \in \widehat{G_m}$.

Now, we only have to prove $\langle \mathbf{d}_i, \mathbf{b}_j \rangle = \delta_{i,j}$ for all $i, j \in G_m \setminus \{1\}$. Since $\mathbf{b}_j = \mathbf{z}_j - \mathbf{z}_1$ for all $j \in G_m \setminus \{1\}$, we have

$$\begin{aligned} \langle \mathbf{d}_i, \mathbf{b}_j \rangle &= (\mathbf{D}^T \mathbf{B})_{i,j} = (\mathbf{D}^T \mathbf{Z} - \mathbf{D}^T \mathbf{Z}_1^M)_{i,j} \\ &= \left(\underbrace{\mathbf{P}_{G_m} \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}}_{=: \mathbf{M}} \underbrace{\mathbf{P}_{G_m}^{-1} - \mathbf{P}_{G_m} \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}}_{=: \mathbf{M}} \mathbf{P}_{G_m}^{-1} \begin{pmatrix} 1 & 1 & \dots & 1 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \right)_{i,j} = \mathbf{M}_{i,j} - \mathbf{M}_{i,1} \end{aligned}$$

for all $i, j \in G_m \setminus \{1\}$. The entry $\mathbf{M}_{i,j}$ of \mathbf{M} is given by $\mathbf{M}_{i,j} = \frac{1}{|G_m|} \sum_{\substack{\chi \in \widehat{G_m} \\ \chi \neq 1}} \chi(i \cdot j^{-1})$. Together with Lemma 2.10 (4) we obtain

$$\mathbf{M}_{i,j} - \mathbf{M}_{i,1} = \frac{1}{|G_m|} \left(\sum_{\substack{\chi \in \widehat{G_m} \\ \chi \neq 1}} \chi(ij^{-1}) - \sum_{\substack{\chi \in \widehat{G_m} \\ \chi \neq 1}} \chi(i) \right) = \frac{1}{|G_m|} \left(\underbrace{\sum_{\chi \in \widehat{G_m}} \chi(ij^{-1})}_{=|G_m|, \text{ if } i=j} - \underbrace{\sum_{\chi \in \widehat{G_m}} \chi(i)}_{=0, \text{ since } i \neq 1} \right) = \delta_{i,j}.$$

By the uniqueness of the dual basis, this implies $\mathbf{b}_j^* = \mathbf{d}_j$ for all $j \in G_m \setminus \{1\}$. Therefore, Theorem 2.13 implies

$$\|\mathbf{b}_j^*\|_2^2 = \|\mathbf{d}_j\|_2^2 = \|\mathbf{d}_1\|_2^2 = |G_m|^{-1} \cdot \sum_{\chi \in \widehat{G_m} \setminus \{1\}} |\lambda_\chi|^{-2}$$

for all $j \in G_m \setminus \{1\}$, since the eigenvalues of \mathbf{D} are given by $0, \frac{1}{\lambda_2}, \dots, \frac{1}{\lambda_n}$ and, again, the components of \mathbf{d}_j are just a permutation of the components of \mathbf{d}_1 . \square

The following theorem summarizes the presented results and provides an upper bound for $\|\mathbf{b}_j^*\|_2$. It can be proven similar to the prime power case in [8, Sect. 3], we only need to bound the new occurring factor $|(1 - \bar{\chi}(p))(1 - \bar{\chi}(q))|$. Therefore we just sketch the proof of the following theorem, for a detailed version see the extended version of this paper [13].

Theorem 4.15. *Let (p, q) be an (α, β) -generator prime pair, and $m := p^\alpha q^\beta$. Then the norm of all \mathbf{b}_j^* for $j \in G_m \setminus \{1\}$ is equal and bounded by*

$$\|\mathbf{b}_j^*\|_2^2 \leq \frac{15C'}{m} + C^2 \log^2(m) \cdot \left(\frac{15\alpha\beta}{2m} + \frac{55(\alpha + \beta)}{8m} + \frac{5\beta}{12p^\alpha} + \frac{5\alpha}{12q^\beta} \right)$$

without the GRH, and

$$\|\mathbf{b}_j^*\|_2^2 \leq C^2(\log \circ \log)^2(m) \cdot \left(\frac{15\alpha\beta}{2m} + \frac{55(\alpha + \beta)}{8m} + \frac{5\beta}{12p^\alpha} + \frac{5\alpha}{12q^\beta} \right),$$

if the GRH holds, for some constants $C, C' > 0$, where C' depends on p, q and C is independent of m . Note that $\log(m) = \alpha \log(p) + \beta \log(q)$ holds for $m = p^\alpha q^\beta$.

Proof. Without loss of generality we just consider the inequality without the GRH. Like in the prime power case, we distinguish between the quadratic and non quadratic characters. Since $\mathbb{Z}_{p^\alpha}^\times$ is cyclic, there is exactly one non-trivial quadratic character of $\mathbb{Z}_{p^\alpha}^\times \cong \widehat{\mathbb{Z}_{p^\alpha}^\times}$ with conductor p . Therefore $\widehat{\mathbb{Z}_m^\times} \cong \widehat{\mathbb{Z}_{p^\alpha}^\times} \times \widehat{\mathbb{Z}_{q^\beta}^\times}$ has only three non-trivial quadratic characters of \mathbb{Z}_m^\times of conductor p, q and pq . Hence, there exists a constant $C' > 0$, such that

$$\sum_{\substack{\chi \in \widehat{G_{p^{l_1}q^{l_2}} \setminus \{1\}} \\ \chi \text{ is quadratic}}} |\lambda_\chi|^{-2} \leq C'$$

for all $l_1, l_2 \in \mathbb{N}$, since the bound of the eigenvalues λ_χ only depends on the conductor f_χ by Theorems 2.15 and 4.13. This implies

$$\begin{aligned} \|\mathbf{b}_j^*\|_2^2 &= |G_m|^{-1} \cdot \left(\sum_{\substack{\chi \in \widehat{G_m} \setminus \{1\} \\ \chi \text{ is quadr.}}} |\lambda_\chi|^{-2} + \sum_{\substack{\chi \in \widehat{G_m} \setminus \{1\} \\ \chi \text{ is not quadr.}}} |\lambda_\chi|^{-2} \right) \\ &\leq \frac{15C'}{m} + \frac{15}{m} \cdot l^2(m) \sum_{\substack{\chi \in \widehat{G_m} \\ \chi \neq 1}} \frac{1}{|(1 - \bar{\chi}(p))(1 - \bar{\chi}(q))|^2 \cdot f_\chi} \end{aligned}$$

with $l(m) := C \log(m) \geq C \log(f_\chi)$ for some constant $C > 0$ by Theorem 2.15. Hence, we have to bound the occurring sum. We split the sum into three sums over the characters with $pq|f_\chi, q \nmid f_\chi$ and $p \nmid f_\chi$. If $pq|f_\chi$, then $|(1 - \bar{\chi}(p))(1 - \bar{\chi}(q))| = 1$, therefore

$$\sum_{\substack{\chi \in \widehat{G_m} \\ pq|f_\chi}} \frac{1}{|(1 - \bar{\chi}(p))(1 - \bar{\chi}(q))|^2 \cdot f_\chi} = \sum_{\substack{\chi \in \widehat{G_m} \\ pq|f_\chi}} \frac{1}{f_\chi} = \sum_{pq|t|m} \frac{1}{t} \sum_{\substack{\chi \in \widehat{G_m} \\ f_\chi=t}} 1 \leq \sum_{pq|t|m} \frac{1}{t} \cdot \frac{t}{2} = \frac{1}{2} \alpha \cdot \beta,$$

where we used that there are at most $|\widehat{G}_t| = \frac{\varphi(t)}{2} \leq \frac{t}{2}$ characters of conductor t in $\widehat{G_m}$.

For the case $q \nmid f_\chi = p^e$ we use the inequality $\sum_{k=1}^{n-1} \frac{1}{|1 - \xi_n^k|^2} \leq 1 + \frac{n}{4} + \frac{1}{9}n^2$, which can be proven by basic analysis, see the extended version of this paper [13]. This and Corollary 4.3 implies

$$\begin{aligned} \sum_{\substack{\chi \in \widehat{G_m} \\ 1 < f_\chi | p^\alpha}} \frac{1}{|(1 - \bar{\chi}(p))(1 - \bar{\chi}(q))|^2 \cdot f_\chi} &\leq \sum_{e=1}^\alpha \frac{1}{p^e} \sum_{\substack{\chi \in \widehat{G_{p^e}} \\ \chi \neq 1}} \frac{1}{|1 - \bar{\chi}(q)|^2} = \sum_{e=1}^\alpha \frac{1}{p^e} \sum_{k=1}^{\frac{\varphi(p^e)}{2} - 1} \frac{1}{|1 - \xi^k \frac{\varphi(p^e)}{2}|^2} \\ &\leq \sum_{e=1}^\alpha \frac{1}{p^e} \cdot \left(1 + \frac{\varphi(p^e)}{8} + \frac{\varphi(p^e)^2}{36} \right) \leq \frac{\alpha}{p} + \frac{\alpha}{8} + \alpha p^{\alpha-2} \frac{(p-1)^2}{36}, \end{aligned}$$

Analogously follows the same bound for the case $p \nmid f_\chi$. Altogether we have

$$\begin{aligned} \|\mathbf{b}_j^*\|_2^2 &\leq \frac{15C_1}{m} + \frac{15}{m} \cdot l^2(m) \left(\frac{\alpha}{p} + \frac{\beta}{q} + \frac{1}{2}\alpha \cdot \beta + \frac{\alpha + \beta}{8} + \beta q^{\beta-2} \frac{(q-1)^2}{36} + \alpha p^{\alpha-2} \frac{(p-1)^2}{36} \right) \\ &\leq \frac{15C_1}{m} + l^2(m) \left(\frac{15\alpha\beta}{2m} + \frac{55(\alpha + \beta)}{8m} + \frac{5\beta}{12p^\alpha} + \frac{5\alpha}{12q^\beta} \right), \end{aligned}$$

where $l(m) = C \log(m)$ for some constant $C > 0$. We have used that $\frac{\alpha}{p} + \frac{\beta}{q} \leq \frac{\alpha}{3} + \frac{\beta}{5} \leq \frac{\alpha+\beta}{3}$. \square

The upper theorem implies $\|\mathbf{b}_j^*\|_2^2 \in O\left(l^3 \cdot \frac{p^l + q^{l+c}}{p^l q^{l+c}}\right)$, where $\alpha = l$ and $\beta = l + c$ for some constant $c \in \mathbb{N}_0$. The following corollary is a direct consequence of this fact and shows, that the basis $\mathbf{b}_1, \dots, \mathbf{b}_k$ for $m = p^\alpha q^\beta$ is well suited for BDD, if (p, q) is a generator prime pair and the distance between α and β is bounded. A proof can be found in the extended version of this paper [13].

Corollary 4.16. *Let (p, q) be a generator prime pair and $c \in \mathbb{N}_0$. Further, let $\alpha_l := l$, $\beta_l := l + c$ and $m_l := p^{\alpha_l} q^{\beta_l}$ for all $l \in \mathbb{N}$. Then $\|\mathbf{b}_j^*\|_2 \rightarrow 0$ for $l \rightarrow \infty$ and all $j \in G_m \setminus \{1\}$ and*

$$m_l \cdot \exp\left(-\frac{1}{8\|\mathbf{b}_j^*\|_2}\right) \rightarrow 0 \text{ for } l \rightarrow \infty.$$

In particular, for every $\omega \in (0, 1)$ Condition 3.5 holds with parameters $M = \|\text{Log}(b_j)^\|_2$ for all $j \in G_m \setminus \{1\}$ and ω for large enough m_l , if the generator $g \in K_{m_l}$ is drawn from a continuous Gaussian.*

References

1. Babai, L.: On Lovász’ lattice reduction and the nearest lattice point problem. *Combinatorica* **6**(1), 1–13 (1986)
2. Bernstein, D.: A subfield-logarithm attack against ideal lattices, February 2014. <http://blog.cr.yp.to/20140213-ideal.html>
3. Biasse, J.-F., Fieker, C.: Subexponential class group and unit group computation in large degree number fields. *LMS J. Comput. Math.* **17**(A), 385–403 (2014)
4. Biasse, J.-F., Song, F.: On the quantum attacks against schemes relying on the hardness of finding a short generator of an ideal in $\mathbb{Q}(\zeta_{p^n})$. Technical report, Tech Report CACR 2015-12 (2015)
5. Biasse, J.-F., Song, F.: Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 893–902. Society for Industrial and Applied Mathematics (2016)
6. Campbell, P., Groves, M., Shepherd, D.: Soliloquy: a cautionary tale. In: ETSI 2nd Quantum-Safe Crypto Workshop, pp. 1–9 (2014)
7. Cohen, H.: *A Course in Computational Algebraic Number Theory*, vol. 4. Springer, Heidelberg (2000)

8. Cramer, R., Ducas, L., Peikert, C., Regev, O.: Recovering short generators of principal ideals in cyclotomic rings. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 559–585. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_20
9. Cramer, R., Ducas, L., Wesolowski, B.: Short Stickelberger class relations and application to Ideal-SVP. Technical report, Cryptology ePrint Archive, Report 2016/885 (2016). <http://eprint.iacr.org/2016/885>
10. Eisenträger, K., Hallgren, S., Kitaev, A., Song, F.: A quantum algorithm for computing the unit group of an arbitrary degree number field. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, pp. 293–302. ACM (2014)
11. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 1–17. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_1
12. Ge, Y.: Elementary properties of cyclotomic polynomials. *Math. Reflect.* **2**, 1–8 (2008)
13. Holzer, P., Wunderer, T., Buchmann, J.: Recovering short generators of principal fractional ideals in cyclotomic fields of conductor $p^\alpha q^\beta$. *IACR Cryptology ePrint Archive* 2017/513 (2017)
14. Ji, C.-G., Lu, H.-W.: Lower bound of real primitive L-function at $s = 1$. *Acta Arith.* **111**, 405–409 (2004)
15. Landau, E.: Über Dirichletsche Reihen mit komplexen Charakteren. *J. für die reine und angewandte Mathematik* **157**, 26–32 (1927)
16. 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
17. 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
18. Micciancio, D., Regev, O.: Lattice-based cryptography. In: Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.) *Post-Quantum Cryptography*, pp. 147–191. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-88702-7_5
19. Mihailescu, P.: Primary cyclotomic units and a proof of Catalan’s conjecture. *J. für die reine und angewandte Mathematik (Crelles Journal)* **572**, 167–195 (2004)
20. Miller, J.C.: Class numbers of real cyclotomic fields of composite conductor. *LMS J. Comput. Math.* **17(A)**, 404–417 (2014)
21. Montgomery, H.L., Vaughan, R.C.: *Multiplicative Number Theory I: Classical Theory*, vol. 97. Cambridge University Press, Cambridge (2006)
22. Neukirch, J., Schappacher, N.: *Algebraic Number Theory. Grundlehren der mathematischen Wissenschaften*. Springer, Heidelberg (1999)
23. Peikert, C., et al.: A decade of lattice cryptography. *Found. Trends® Theor. Comput. Sci.* **10(4)**, 283–424 (2016)
24. Schoof, R.: *Catalan’s Conjecture*. Springer Science & Business Media, Heidelberg (2010)
25. Siegel, C.: Über die Classenzahl quadratischer Zahlkörper. *Acta Arith.* **1(1)**, 83–86 (1935)
26. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_25
27. Washington, L.C.: *Introduction to Cyclotomic Fields*, 2nd edn. Springer, Heidelberg (1996)

Revisiting a Masked Lookup-Table Compression Scheme

Srinivas Vivek^(✉) 

University of Bristol, Bristol, UK
sv.venkatesh@bristol.ac.uk

Abstract. Lookup-table based side-channel countermeasure is the prime choice for masked S-box software implementations at very low orders. To mask an n -bit to m -bit S-box at first- and second- orders, one requires a temporary table in RAM of size $m \cdot 2^n$ bits. Recently, Vadnala (CT-RSA 2017) suggested masked table compression schemes at first- and second-orders to reduce the table size by (approximately) a factor of 2^l , where l is a parameter. Though greater compression results in a greater execution time, these proposals would still be attractive for highly resource constrained devices.

In this work, we contradict the second-order security claim of the second-order table compression scheme by Vadnala. We do this by exhibiting several pairs of intermediate variables that jointly depend on the bits of the secret. Motivated by the fact that randomness is also a costly resource for highly resource constrained devices, we then propose a variant of the first-order table compression scheme of Vadnala that has the new randomness complexity of about l instead of 2^l for the original proposal. We achieve this without inducing any noticeable difference in the overall execution time or memory requirement of the original scheme. Finally, we show that the randomness complexity of l is optimal in an algebraic sense.

Keywords: Side-channel attack · Masking · Block cipher Implementation

1 Introduction

Side-channel attacks on cryptographic implementations exploit physical characteristics of an execution such as timing, power consumption or electromagnetic emission pattern, to name but a few [Koc96, KJJ99]. Block cipher implementations have been a major target for these attacks. Over the years, a number of countermeasures against these attacks have been developed too. Of these, (boolean) *masking* is one of the very first and still a popular technique to protect

This work has been supported in part by the European Union's H2020 Programme under grant agreement number ICT-644209 (HEAT), and by the EPSRC under grant number EP/M016803/1.

block cipher implementations [CJRR99, ISW03]. The basic idea behind masking is to split every sensitive variable into one or more shares and process them in such a way that intermediate variables do not reveal information about these sensitive variables. An implementation is said to be secure against t -th order attacks in the *probing leakage model* if any subset of t intermediate variables (including the input and output shares) jointly is statistically independent of the secret variables [ISW03]. Hence at least $t + 1$ shares are needed for each of the secret inputs to achieve t -th order security. As the number of shares increase so does the complexity of the side-channel attacks [CJRR99, PR13, DDF14]. Moreover, since the processing of affine functions is straightforward, the main challenge in masking block ciphers is the masking of the non-linear operations, in particular, the S-box functions.

Recent years have witnessed an increased focus on the design and improvement of higher-order (boolean/arithmetic) masking schemes for S-boxes. For instance, see [CGPZ16, GR16, PV16, CRZ17, GR17, GRVV17, JS17] and the references within. These masking schemes can roughly be categorised into polynomial/ arithmetic-circuit based masking schemes (including the bit-sliced masking technique) on one hand, and look-up table-based masking schemes on the other. As the above works have shown, at higher orders, polynomial-based schemes have been more efficient than table-based schemes in terms of time, memory and randomness complexity. In spite of the above advancement of polynomial-based masking schemes, at very low orders, such as first- and second-orders, table-based masking schemes are the most effective due to low overheads. Unsurprisingly, vast majority of the commercial implementations opt just for first- or second-order masked implementations due to efficiency concerns.

Original Look-up Table-based Masking. The original table-based first-order masking of an (n, m) -S-box S consists of creating a temporary table $T : \{0, 1\}^n \rightarrow \{0, 1\}^m$ in RAM [CJRR99]:

$$T(a) = S(x_1 \oplus a) \oplus y_1 \quad \forall 0 \leq a \leq 2^n - 1,$$

where x_1 and x_2 are the input shares such that the secret $x = x_1 \oplus x_2 \in \{0, 1\}^n$, and y_1 and y_2 are the output shares such that $S(x) = y_1 \oplus y_2 \in \{0, 1\}^m$. Using T , y_2 can simply be computed as $y_2 = T(x_2)$. The RAM memory requirement for the table T is $m \cdot 2^n$ bits.

Prouff and Rivain [PR07] suggested a first-order S-box masking scheme that mainly requires only two m -bit registers, hence doing away with the need to store the table T . Though this method only requires (essentially) a constant amount of memory, the overhead induced is a factor of about 30–35, while it is just 2 - 3 for the original method [Vad17]. Moreover, in the original method, the table T can be computed “offline” hence significantly reducing the “online” computation time. But in the method of [PR07], the whole table is computed (on the fly) during the online phase and hence the relatively large overhead.

The second-order S-box masking schemes of Schramm and Paar [SP06], and Rivain, Dottax and Prouff [RDP08] also require a temporary table of size $m \cdot 2^n$

bits, while the scheme of Coron [Cor14] requires $3 \cdot m \cdot 2^n$ bits [CRZ17]. The authors of [RDP08] also suggest a second-order scheme that requires only two m -bit registers and 2^n bits of RAM memory.

Masked Table Compression. In order to reduce the RAM memory requirement for highly resource constrained devices, Rao *et al.* [RRST02] suggested a table compression scheme for first-order masking that requires only $\approx m \cdot 2^{n-1}$ bits for the temporary table. For the case of AES, one now needs only 128 bytes for the table instead of 256 bytes for the original table-based method. In general, one can improve the memory complexity of the method of [RRST02] to $\approx m \cdot 2^n / l$ bits, for a parameter l such that $1 \leq l \leq m$.

Inspired by the method of [RRST02], Vadnala [Vad17] suggested masked table compression techniques that achieve better compression for both the first- and second-order S-box masking. It is shown that the memory requirement for the first-order case can be reduced to $\approx m \cdot 2^{n-l} + (n-l) \cdot 2^l$ bits, where l is a parameter, called *compression level*, such that $1 \leq l \leq n$. For the second-order case, it is shown to be $\approx m \cdot 2^{n-l} + (n-l+1) \cdot 2^l$ bits. The author also investigated the (online-)time and (RAM) memory trade-off in between the two extremes mentioned above for the original method. Reasonably efficient first- and second-order masked implementations of AES-128 were obtained using only about 40 bytes of RAM memory [Vad17]. The proposed schemes were argued to be secure in the probing leakage model [ISW03].

Let us very briefly illustrate the technique of [Vad17] for the first-order case. The main idea is to “pack” 2^l table entries of the original randomised table into a single entry of table T_1 :

$$T_1(a^{(1)}) = \left(\bigoplus_{0 \leq i \leq 2^l - 1} S((a^{(1)} \oplus r_i) \parallel i) \right) \oplus y_1, \quad \forall 0 \leq a^{(1)} \leq 2^{n-l} - 1,$$

where $r_i \in \{0, 1\}^{n-l}$ are uniform random and independently sampled. One needs to carefully access this table to produce another table (that need not be stored) which is then securely accessed with the shares of the remaining l -bits of the secret x (cf. Sect. 2.1). Note that for the given compression level l , one needs to make 2^l calls to a random number generator to generate the r_i s.

1.1 Our Contribution

We *contradict* the second-order security claim of the second-order masked table compression scheme(s) of [Vad17]. We exhibit a second-order attack on the scheme(s) by demonstrating the existence of several pairs of intermediate variables that jointly depend on the secret (cf. Lemma 1). Our attack is, in spirit, similar to the third-order attack suggested by Coron, *et al.* [CPR07] on the higher-order masking scheme of Schramm and Paar [SP06].

Motivated by the fact that the generation of quality randomness is possibly a costly operation on highly resource constrained devices, we then revisit the first-order scheme(s) of [Vad17] and propose a variant (first-order) scheme(s)

that requires only about l calls to a random number generator instead of 2^l required for the original scheme(s) (cf. Sect. 2.2). Our main idea is to generate the 2^l values r_i using only $l + 1$ random values $\gamma_0, \dots, \gamma_l \in \{0, 1\}^{n-l}$. We do this by setting r_i to a subset (xor) sum of the γ_j s. Other than this difference, the rest of our method is the same as in [Vad17]. This implies that apart from the difference in the time to compute the r_i s at the very beginning, there is hardly any noticeable difference in the time and memory complexity of our method and that of [Vad17]. We too prove the first-order security of our scheme in the probing leakage model (cf. Theorem 1). It may be noted that it is straightforward to securely compose first-order secure schemes in a bigger construction.

Finally, we show that the randomness complexity of l that we achieved is (nearly) *optimal* in an algebraic sense (cf. Sect. 2.3). Specifically, our computation model assumes that the only arithmetic operations allowed are *xors*, i.e., \mathbb{F}_2 -linear operations. This is a reasonable assumption since nearly all the known table-based masking schemes satisfy this assumption [CJRR99, RRST02, SP06, PR07, RDP08, Cor14, CRZ17, Vad17].

Organisation of the Paper. To gradually introduce the techniques of masked table compression, we first describe our contributions for the first-order case in Sect. 2 before presenting our attack on the second-order scheme in Sect. 3.

2 Improved First-Order Table Compression Scheme

2.1 Original First-Order Scheme

Before we present our improved first-order masked table compression scheme, let us first briefly recollect the original first-order proposal from [Vad17, Sect. 2]. The notation we use here is somewhat different from that in [Vad17, Sect. 2], and we summarise the changes in Remark 1. Throughout the paper, by $b \stackrel{\$}{\leftarrow} \{0, 1\}^k$ we denote a uniform random and independent sampling of a k -bit string.

Consider an (n, m) -S-box $S : \{0, 1\}^n \rightarrow \{0, 1\}^m$, where $n \geq m$. The task is to securely evaluate $S(x)$, given the input shares $x_1 \stackrel{\$}{\leftarrow} \{0, 1\}^n$ and x_2 such that $x = x_1 \oplus x_2 \in \{0, 1\}^n$, ensuring that no intermediate variable is statistically dependent on the “secret” x . The outputs are two shares $y_1 \stackrel{\$}{\leftarrow} \{0, 1\}^m$ and y_2 such that $S(x) = y_1 \oplus y_2 \in \{0, 1\}^m$.

Let $1 \leq l \leq n$ be the compression level. Define the functions $S_i : \{0, 1\}^{n-l} \rightarrow \{0, 1\}^m$ ($0 \leq i \leq 2^l - 1$) as

$$S_i(a^{(1)}) := S(a^{(1)}||i), \quad \forall 0 \leq a^{(1)} \leq 2^{n-l} - 1, \tag{1}$$

where i is represented using l bits. The main idea in [Vad17] is to “pack” 2^l table entries of the original randomised table into a single entry of table T_1 . More precisely, let

$$T_1(a^{(1)}) = \left(\bigoplus_{0 \leq i \leq 2^l - 1} S_i(a^{(1)} \oplus r_i) \right) \oplus y_1, \quad \forall 0 \leq a^{(1)} \leq 2^{n-l} - 1, \tag{2}$$

where

$$r_i \stackrel{\$}{\leftarrow} \{0, 1\}^{n-l}, \quad \forall 0 \leq i \leq 2^l - 1, \quad (3)$$

are uniform random and independently sampled.

Let

$$x =: x^{(1)} \parallel x^{(2)}, \quad \text{where } x^{(1)} \in \{0, 1\}^{n-l}, \quad x^{(2)} \in \{0, 1\}^l. \quad (4)$$

Similarly, let

$$x_1 =: x_1^{(1)} \parallel x_1^{(2)}, \quad \text{where } x_1^{(1)} \in \{0, 1\}^{n-l}, \quad x_1^{(2)} \in \{0, 1\}^l, \quad (5)$$

and

$$x_2 =: x_2^{(1)} \parallel x_2^{(2)}, \quad \text{where } x_2^{(1)} \in \{0, 1\}^{n-l}, \quad x_2^{(2)} \in \{0, 1\}^l. \quad (6)$$

In [Vad17, Sect. 2] it is mentioned that $x_1^{(1)} = \bigoplus_{0 \leq i \leq 2^l - 1} r_i$. But this is not a necessity and we assume that x_1 is independently chosen. The next step is to compute a table $U : \{0, 1\}^l \rightarrow \{0, 1\}^m$ comprising of all the values $S_i(x^{(1)}) \oplus y_1$, where $0 \leq i \leq 2^l - 1$, by securely accessing the tables T_1 and S_i as follows:

$$S_i(x^{(1)}) \oplus y_1 = T_1((x_1^{(1)} \oplus r_i) \oplus x_2^{(1)}) \oplus \bigoplus_{0 \leq j \leq 2^l - 1, j \neq i} S_j(((x_1^{(1)} \oplus r_i) \oplus x_2^{(1)}) \oplus r_j).$$

For security considerations, the expression inside the parentheses above and elsewhere must be evaluated with higher precedence. To compute the second output share $y_2 = S(x) \oplus y_1$, the table U needs to be accessed at $x^{(2)}$. But if one directly accesses the table U as mentioned, then it would leak l -bits of the secret x . Therefore, *instead* of creating the table U , a randomised table T_2 corresponding to U shifted by $x_1^{(2)}$ is created as follows:

$$T_2(a^{(2)}) = T_1((x_1^{(1)} \oplus r_{(a^{(2)} \oplus x_1^{(2)})}) \oplus x_2^{(1)}) \oplus \bigoplus_{0 \leq j \leq 2^l - 1, j \neq (a^{(2)} \oplus x_1^{(2)})} S_j(((x_1^{(1)} \oplus r_{(a^{(2)} \oplus x_1^{(2)})}) \oplus x_2^{(1)}) \oplus r_j), \quad (7)$$

where $0 \leq a^{(2)} \leq 2^l - 1$. Finally, compute

$$y_2 = T_2(x^{(2)}).$$

The above scheme is proven to be first-order secure in the probing leakage model [ISW03]. Namely, every intermediate variable (including the input and output shares) is shown to be independent of the secret x . Note that the table T_1 can be computed offline. As the value of the compression level l increases, then so does the online computation time. The table T_1 has 2^{n-l} m -bit entries, while the table T_2 has 2^l m -bit entries. Hence the combined size of the two

tables is $(2^{n-l} + 2^l) \cdot m$ bits compared to the $2^n \cdot m$ bits needed for the original randomised table-based (first-order) masking scheme. But in the above scheme we now need to also store the 2^l random values r_i each $n - l$ bits long.

It is suggested in [Vad17] how to do away with the need to store the table T_2 . This is based on a first-order S-box masking scheme from [PR07] that mainly uses only two registers (instead of a table) to compute the output shares y_1 and y_2 . The only (implicit) requirement to apply the method of [PR07] in different contexts is that random access must be possible for the table that is being masked. Since the entries of the table T_2 can be computed in any arbitrary order, one can straightforwardly apply the technique of [PR07] in the current context. Hence the RAM memory complexity of the first-order table compression scheme from [Vad17] is approximately $2^{n-l} \cdot m + 2^l \cdot (n - l)$ bits.

Remark 1. The variables $x_1, x_1^{(1)}, x_1^{(2)}, x_2, x_2^{(1)}, x_2^{(2)}, y_1$, and $a^{(1)}$, in this section correspond to, respectively, $r, r^{(1)}, r^{(2)}, x_1, x_1^{(1)}, x_1^{(2)}, s$, and u , in [Vad17, Sect. 2]. The final step that computes y_2 is also slightly different compared to [Vad17].

2.2 Our Method

Our main idea to reduce the randomness complexity of the first-order scheme from [Vad17] is as follows. Instead of choosing 2^l random values $r_i \stackrel{\$}{\leftarrow} \{0, 1\}^{n-l}$ (cf. (3)), we compute the required r_i using only $l + 1$ random values

$$\gamma_j \stackrel{\$}{\leftarrow} \{0, 1\}^{n-l} \quad \forall 0 \leq j \leq l$$

by xoring different subsets of this smaller set of random values. By

$$\text{bits}_l(i) \in \mathbb{F}_2^l \quad \forall 1 \leq i \leq 2^l - 1$$

we mean an l -bit vector consisting of the bits in the binary representation of i . Let $\text{bits}_l(i)[0]$ denote the least significant bit and, consequently, $\text{bits}_l(i)[l - 1]$ denotes the most significant bit of i (which could possibly be 0). Define

$$\begin{aligned} r_0 &:= \gamma_l, \\ r_i &:= \sum_{j=0}^{l-1} \text{bits}_l(i)[j] \cdot \gamma_j, \quad \forall 1 \leq i \leq 2^l - 1. \end{aligned} \tag{8}$$

Hence each of r_1, \dots, r_{2^l-1} is computed as the xor of the subset of γ_j s defined by the binary representation of their indices. When $i = 0$, the subset xor of γ_j s is zero. Hence r_0 is set to a fresh random value. This procedure is summarised in Algorithm 1.

Once the values r_i are generated and stored, then the rest of the procedure is the same as in the original scheme recollected in Sect. 2.1 (but also see Remark 2). Hence the proof of correctness for our improved method follows automatically. For completeness, we summarise the complete (improved) first-order masked table compression scheme in Algorithm 4.

Algorithm 1. Computing $\{r_0, \dots, r_{2^l-1}\}$ according to (8).

```

Input:  $\gamma_0, \dots, \gamma_l \in \{0, 1\}^{n-l}$ .
Output:  $r_0, \dots, r_{2^l-1} \in \{0, 1\}^{n-l}$ .
1:  $r_0 \leftarrow \gamma_l$ 
2: for  $i \leftarrow 1$  to  $2^l - 1$  do
3:    $r_i \leftarrow 0$ 
4:   for  $j \leftarrow 0$  to  $l - 1$  do
5:     if  $\text{bits}_l(i)[j] \neq 0$  then
6:        $r_i \leftarrow r_i \oplus \gamma_j$ 
7:     end if
8:   end for
9: end for
10: return  $r_0, \dots, r_{2^l-1}$ 

```

Algorithm 2. Computing table T_1 for first-order masked table compression (cf. (2)).

```

Input:  $(n, m)$ -S-box table  $S$ , an output share  $y_1 \in \{0, 1\}^m$ ,  $\{r_0, \dots, r_{2^l-1}\}$  from Algorithm 1.
Output: Table  $T_1$ .
1: define  $S_i(a^{(1)}) := S(a^{(1)} || i)$  (cf. (1))
2: for  $a^{(1)} \leftarrow 0$  to  $2^{n-l} - 1$  do
3:    $z \leftarrow y_1$ 
4:   for  $i \leftarrow 0$  to  $2^l - 1$  do
5:      $z \leftarrow z \oplus S_i(a^{(1)} \oplus r_i)$ 
6:   end for
7:    $T_1(a^{(1)}) \leftarrow z$ 
8: end for
9: return  $T_1$ 

```

Remark 2. In Algorithm 3, the variable z is initialised to $T_1(\text{ind}_1)$ and then xored with $\bigoplus S_j(\text{ind}_2)$. But in [Vad17, Sect. 2], the variable z is initialised to 0 and the above xor was computed at the end. The latter approach could lead to a first-order security flaw for our method due to the “random” values r_i being related in our method.

Remark 3. The execution time for our method (Algorithm 4) and that of [Vad17] is the same except for the time required to generate the values r_i . In the latter method it requires 2^l calls to the random number generator, while for our method it needs only $l + 1$ calls plus the computation of $l \cdot 2^{l-1}$ xors.

Remark 4. The RAM memory complexity is the same for both our method and for [Vad17] since the extra variables γ_j in our method can be discarded right after computing and storing the values r_i . The RAM memory complexity for both the methods is approximately $2^{n-l} \cdot m + 2^l \cdot (n - l)$ bits (in spite of computing the table T_2 on the fly).

Algorithm 3. Computing table T_2 for first-order masked table compression (cf. (7)).

Input: (n, m) -S-box table S , two input shares x_1 and x_2 such that $x_1 \oplus x_2 = x \in \{0, 1\}^n$, an output share $y_1 \in \{0, 1\}^m$, $\{r_0, \dots, r_{2^l-1}\}$ from Algorithm 1, table T_1 from Algorithm 2.

Output: Table T_2 .

```

1: define  $S_i(a^{(1)}) := S(a^{(1)}||i)$  (cf. (1)),  $x_1 =: x_1^{(1)} || x_1^{(2)}$  (cf. (5)),  $x_2 =: x_2^{(1)} || x_2^{(2)}$ 
   (cf. (6))
2: for  $a^{(2)} \leftarrow 0$  to  $2^l - 1$  do
3:    $k \leftarrow a^{(2)} \oplus x_1^{(2)}$ 
4:    $ind_1 \leftarrow (x_1^{(1)} \oplus r_k) \oplus x_2^{(1)}$ 
5:    $z \leftarrow T_1[ind_1]$ 
6:   for  $j \leftarrow 0$  to  $2^l - 1$  do
7:     if  $j \neq k$  then
8:        $ind_2 \leftarrow ind_1 \oplus r_j$ 
9:        $z \leftarrow z \oplus S_j(ind_2)$ 
10:    end if
11:  end for
12:   $T_2(a^{(2)}) \leftarrow z$ 
13: end for
14: return  $T_2$ 

```

Remark 5. The randomness complexity of our method in terms of the number of calls to a random number generator is $l + 3$ instead of $2^l + 2$ for [Vad17]. In terms of the number of random bits generated, it is $(l + 1) \cdot (n - l) + n + m$ for ours instead of $2^l \cdot (n - l) + n + m$ for [Vad17].

The above complexity estimates are exclusively for the masked computation of a single S-box and hence does *not* include the processing of the full cipher. We refer to [Vad17, Sect. 4] for a concrete performance evaluation of masked AES-128 on a 32-bit ARM Cortex-M3 based micro-controller. We expect that on such relatively big architectures the execution times for our method and that of [Vad17] will not differ significantly.

Theorem 1. *Algorithm 4 is first-order secure in the probing leakage model.*

Security Proof. To this end, we just need to show that every intermediate variable is independent of the secret input x . It is obvious that all the intermediate values appearing until (including) Step 7 of Algorithm 4 are independent of x since they can be computed offline. These intermediate variables can simply be simulated by picking suitable random values. Out of the intermediate variables occurring in the remaining Steps 8, 9 and 10 of Algorithm 4, the variables $x_2^{(2)} = x^{(2)} \oplus x_1^{(2)}$, y_1 , and $y_2 = T_2(x_2^{(2)}) = S(x) \oplus y_1$ in Steps 9 and 10 are clearly independent of x . This leaves us to deal with only the variables (including the inputs and outputs) occurring in the computation of table T_2 in Algorithm 3.

The (probability distribution of the) variable $k = a^{(2)} \oplus x_1^{(2)}$ is uniform random and independent of x due to x_1 . Since each $r_i \in \{0, 1\}^{n-l}$ ($0 \leq i \leq 2^l - 1$)

Algorithm 4. Improved first-order masked table compression

Input: (n, m) -S-box table S , two input shares x_1 and x_2 such that $x_1 \oplus x_2 = x \in \{0, 1\}^n$.

Output: Two output shares y_1 and y_2 such that $y_1 \oplus y_2 = S(x) \in \{0, 1\}^m$.

- 1: **define** $x_2 =: x_2^{(1)} \parallel x_2^{(2)}$ (cf. (6))
- 2: $y_1 \leftarrow \{0, 1\}^m$
- 3: **for** $j \leftarrow 0$ **to** l **do**
- 4: $\gamma_j \stackrel{\$}{\leftarrow} \{0, 1\}^{n-l}$
- 5: **end for**
- 6: Compute $r_0, \dots, r_{2^l-1} \leftarrow$ Algorithm 1 ($\gamma_0, \dots, \gamma_l$)
- 7: Compute table $T_1 \leftarrow$ Algorithm 2 ($S, y_1, \{r_0, \dots, r_{2^l-1}\}$)
 { All the above steps may be computed offline. }
- 8: Compute table $T_2 \leftarrow$ Algorithm 3 ($S, x_1, x_2, y_1, \{r_0, \dots, r_{2^l-1}\}, T_1$)
- 9: $y_2 \leftarrow T_2(x_2^{(2)})$
- 10: **return** y_1, y_2

is uniform random and independent of x and x_1 (because r_i s are xors of uniform random and independent γ_j s), the variables $x_1^{(1)} \oplus r_k$ and $ind_1 = x^{(1)} \oplus r_k$ are uniform random and independent of x . Because each entry of the table T_1 is masked with y_1 , hence they too are uniform random and independent of x . This implies that the initial value of $z = T_1(ind_1)$ is also uniform random and independent of x .

Consider the values assumed by the variable $ind_2 = x^{(1)} \oplus r_k \oplus r_j$. Since $j \neq k$ and if $j, k \neq 0$, it is easy to see that $r_k \oplus r_j = r_{k \oplus j}$. Since all the r_i s are uniform random and independent of x , so is ind_2 . If $j = 0$, then $ind_2 = x^{(1)} \oplus r_k \oplus \gamma_l$, and if $k = 0$, then $ind_2 = x^{(1)} \oplus \gamma_l \oplus r_j$, and the above conclusion follows easily. This also means that $S_j(ind_2)$ occurring in Step 9 of Algorithm 3 is also independent of x . Finally, we need to show that all the values of z from Step 9 (including the Step 12) are independent of x . As reasoned above, the initial value of $z = T_1(ind_1)$ is masked with y_1 . Since each of the values $S_j(ind_2)$ is also independent of y_1 , this implies that all the values assumed by z are always uniform random and independent of x . This completes the security proof. \square

2.3 Lower Bound on Randomness Complexity

We next show that the randomness complexity of our method from Sect. 2.2 has (nearly) optimal randomness complexity in an algebraic sense. Precisely, we prove that one needs to make at least l calls to a random number generator to compute the values $r_0, \dots, r_{2^l-1} \in \{0, 1\}^{n-l}$ used to compute table T_2 (cf. Remark 5). Needless to say, this lower bound is also applicable to the original scheme from [Vad17, Sect. 2]. To prove our lower bound, we assume that the only arithmetic operations performed are xors, i.e., only \mathbb{F}_2 -linear operations, which indeed is a typical scenario for table-based masking schemes.

Theorem 2. *Algorithm 4 needs at least $l \cdot (n - l)$ uniform randomly generated bits to be first-order secure in the probing leakage model if only \mathbb{F}_2 -linear operations are performed.*

Proof. Let us assume that fewer than l calls are made to the random number generator to compute the r_i s. Then we will exhibit an intermediate value that depends on the secret x . Let the generated random values be $\nu_1, \dots, \nu_t \in \{0, 1\}^{n-l}$, where $t \leq l - 1$. In the assumed computation model, all the computed values r_0, \dots, r_{2^l-1} will be of the form

$$r_i = c_i \bigoplus_{1 \leq j \leq t} b_j \cdot \nu_j, \quad \text{where } b_j \in \mathbb{F}_2, c_i \in \{0, 1\}^{n-l},$$

for $0 \leq i \leq 2^l - 1$. This implies that there exist some r_p and r_q ($p \neq q$) such that

$$r_p \oplus r_q = c_p \oplus c_q$$

is a constant. The variable $ind_2 = x^{(1)} \oplus c_p \oplus c_q$ in Step 8 of Algorithm 3 when $a^{(2)} = p$ and $j = q$. Hence this value is correlated with the $n - l$ most significant bits of x . This proves our claim. \square

3 Attack on the Second-Order Masked Table Compression Method of [Vad17]

3.1 Original Second-Order Scheme

Before we present our attack on the second-order masked table compression scheme from [Vad17, Sect. 3], let us first recollect the original scheme. To be consistent with the notation in Sect. 2, we will use a slightly different notation here than that in [Vad17, Sect. 3], and we summarise the changes in Remark 6.

The second-order table compression scheme from [Vad17] is based on the second-order S-box masking scheme from [RDP08, Sect. 3.1]. Consider again an (n, m) -S-box $S : \{0, 1\}^n \rightarrow \{0, 1\}^m$, where $n \geq m$. On input three shares $x_1 \stackrel{\$}{\leftarrow} \{0, 1\}^n$, $x_2 \stackrel{\$}{\leftarrow} \{0, 1\}^n$ and x_3 such that $x = x_1 \oplus x_2 \oplus x_3 \in \{0, 1\}^n$, the task is to compute the three output shares $y_1 \stackrel{\$}{\leftarrow} \{0, 1\}^m$, $y_2 \stackrel{\$}{\leftarrow} \{0, 1\}^m$ and y_3 such that $S(x) = y_1 \oplus y_2 \oplus y_3 \in \{0, 1\}^m$. In order for the scheme to be second-order secure in the probing model, the requirement is that the joint probability distribution of any pair of intermediate variables (including the input and the output shares) is statistically independent of the secret x . At a high level the main technique behind the second-order table compression scheme is similar to that of the first-order compression scheme presented in Sect. 2. First, create a table T_1 that “packs” 2^l randomised S-box values:

$$T_1(b^{(1)}) := \left(\left(\bigoplus_{0 \leq i \leq 2^l-1} S_i(x_3^{(1)} \oplus a^{(1)} \oplus r_i) \right) \oplus y_1 \right) \oplus y_2, \tag{9}$$

where

$$b^{(1)} = a^{(1)} \oplus ((x_1^{(1)} \oplus v^{(1)}) \oplus x_2^{(1)}), \tag{10}$$

for all $0 \leq a^{(1)} \leq 2^{n-l} - 1$, $S_i(a^{(1)}) := S(a^{(1)} || i)$ as in (1), $v^{(1)} \stackrel{\$}{\leftarrow} \{0, 1\}^{n-l}$, and $r_i \stackrel{\$}{\leftarrow} \{0, 1\}^{n-l}$ as in (3).

Let us recollect the notations $x =: x^{(1)} || x^{(2)}$, $x_1 =: x_1^{(1)} || x_1^{(2)}$, $x_2 =: x_2^{(1)} || x_2^{(2)}$ from (4) to (6). Similarly, let $x_3 =: x_3^{(1)} || x_3^{(2)}$. The next step is to compute a table $U : \{0, 1\}^l \rightarrow \{0, 1\}^m$ consisting of the values $S_i(x^{(1)}) \oplus y_1$, where $0 \leq i \leq 2^l - 1$, by carefully accessing the tables T_1 and S_i . We have

$$S_i(x^{(1)}) \oplus y_1 \oplus y_2 = T_1(v^{(1)} \oplus r_i) \oplus \bigoplus_{0 \leq j \leq 2^l - 1, j \neq i} S_j(x^{(1)} \oplus r_i \oplus r_j).$$

Now the final output share $y_3 = S(x) \oplus y_1 \oplus y_2$ can be computed by accessing the table U at $x^{(2)}$. Of course, this cannot be done as $x^{(2)}$ must never be computed explicitly. *Instead* of computing the table U , a second-order masked table T_2 is created that is accessed with the shares of $x^{(2)}$.

Let

$$T_2(b^{(2)}) := T_1(v^{(1)} \oplus r_{(x_3^{(2)} \oplus a^{(2)})}) \oplus \bigoplus_{0 \leq j \leq 2^l - 1, j \neq a^{(2)}} S_{(x_3^{(2)} \oplus j)}(x^{(1)} \oplus r_{(x_3^{(2)} \oplus a^{(2)})} \oplus r_{(x_3^{(2)} \oplus j)}), \tag{11}$$

for all $0 \leq a^{(2)} \leq 2^l - 1$, where

$$b^{(2)} := a^{(2)} \oplus ((x_1^{(2)} \oplus v^{(2)}) \oplus x_2^{(2)}), \tag{12}$$

and $v^{(2)} \stackrel{\$}{\leftarrow} \{0, 1\}^l$. Once the table T_2 is computed, the output share y_3 can simply be computed as

$$y_3 = T_2(v^{(2)}).$$

We will not recollect here the exact details of how the tables T_1 and T_2 are computed since it is not necessary to present our attack. We refer to [Vad17, Algorithm 8] for these details. As observed in [Vad17], it is not necessary to store the table T_2 . Instead, it can be computed “on the fly” by making use of the technique from [RDP08, Algorithm 3].

Remark 6. The variables $x^{(1)}$, $x^{(2)}$, $x_1^{(1)}$, $x_1^{(2)}$, $x_2^{(1)}$, $x_2^{(2)}$, x_3 , $x_3^{(1)}$, $x_3^{(2)}$, y_1 , y_2 , $v^{(1)}$, and $v^{(2)}$, in this section correspond to, respectively, y , b , y_1 , b_1 , y_2 , b_2 , x' , y' , b' , s_1 , s_2 , y_3 , and b_3 , in [Vad17, Sect. 3]. The pairs of variables $(a^{(1)}, b^{(1)})$ and $(a^{(2)}, b^{(2)})$ in our description both correspond, in different contexts, to (a, a') in [Vad17, Sect. 3].

3.2 Our Attack

We now present a second-order security flaw in the second-order masked table compression scheme from [Vad17, Sect. 3]. Our attack is, in spirit, similar to the third-order attack suggested in [CPR07] on the higher-order masking scheme of [SP06]. More precisely, we prove the following lemma that establishes the existence of many pairs of intermediate variables that jointly depend on the bits of the secret x .

Lemma 1. *Let $\beta_1, \beta_2 \in \{0, 1\}^l$. Then*

$$T_2(\beta_1) \oplus T_2(\beta_2) = S_{(\beta_1 \oplus x^{(2)} \oplus v^{(2)})}(x^{(1)}) \oplus S_{(\beta_2 \oplus x^{(2)} \oplus v^{(2)})}(x^{(1)}).$$

Proof. From (11), we have

$$T_2(\beta_1) = T_1(v^{(1)} \oplus r_{(x_3^{(2)} \oplus \alpha_1^{(2)})}) \oplus \bigoplus_{0 \leq j \leq 2^l - 1, j \neq \alpha_1^{(2)}} S_{(x_3^{(2)} \oplus j)}(x^{(1)} \oplus r_{(x_3^{(2)} \oplus \alpha_1^{(2)})} \oplus r_{(x_3^{(2)} \oplus j)}), \quad (13)$$

where, from (12),

$$\alpha_1^{(2)} := \beta_1 \oplus ((x_1^{(2)} \oplus v^{(2)}) \oplus x_2^{(2)}). \quad (14)$$

From (9), (10) and (14), we have

$$T_1(v^{(1)} \oplus r_{(x_3^{(2)} \oplus \alpha_1^{(2)})}) = y_1 \oplus y_2 \oplus \bigoplus_{0 \leq j \leq 2^l - 1} S_j(x^{(1)} \oplus r_{\beta_1 \oplus x^{(2)} \oplus v^{(2)}} \oplus r_j).$$

From (14) and by a change of index, we obtain

$$T_1(v^{(1)} \oplus r_{(x_3^{(2)} \oplus \alpha_1^{(2)})}) = y_1 \oplus y_2 \oplus S_{(\beta_1 \oplus x^{(2)} \oplus v^{(2)})}(x^{(1)}) \oplus \bigoplus_{0 \leq j \leq 2^l - 1, j \neq \alpha_1^{(2)}} S_{(x_3^{(2)} \oplus j)}(x^{(1)} \oplus r_{(x_3^{(2)} \oplus \alpha_1^{(2)})} \oplus r_{(x_3^{(2)} \oplus j)}).$$

On substituting the above equation in (13), we get

$$T_2(\beta_1) = S_{(\beta_1 \oplus x^{(2)} \oplus v^{(2)})}(x^{(1)}) \oplus y_1 \oplus y_2.$$

Similarly, we obtain

$$T_2(\beta_2) = S_{(\beta_2 \oplus x^{(2)} \oplus v^{(2)})}(x^{(1)}) \oplus y_1 \oplus y_2.$$

Finally,

$$T_2(\beta_1) \oplus T_2(\beta_2) = S_{(\beta_1 \oplus x^{(2)} \oplus v^{(2)})}(x^{(1)}) \oplus S_{(\beta_2 \oplus x^{(2)} \oplus v^{(2)})}(x^{(1)}).$$

This proves the claim. □

The above result suggests that *every* pair of values in the table T_2 *jointly* depends on $n - l$ bits of the secret x . In particular, if the compression level $l = 1$, this means that each pair of values will jointly “leak” all but one bit of the secret.

Remark 7. Our attack only exploits the values in the table T_2 and not the means by which it is computed. Hence our attack is also applicable to the variant scheme in [Vad17, Sect. 3] where T_2 is not stored but only computed on the fly.

Remark 8. For our attack the compression level can be any value l such that $1 \leq l \leq n - 1$. Note that our attack is not applicable when $l = 0$, which corresponds to the scheme from [RDP08, Sect. 3.1], and when $l = n$. Our attack also does not work for those functions S that depend only on the least significant l bits of its input as this part of the input is randomised. But such functions are of little interest for use as cryptographic S-boxes.

Remark 9. In side-channel experiments one hardly gets the values of intermediate variables without any error. Instead, a noisy function of the bits is observed, for e.g., noisy Hamming weight values. We refer to the techniques in [PRB09, SVO+10] to extract the bits of the secret from the noisy experimental data.

Acknowledgements. We would like to thank Srinivas Karthik and Yan Yan for helpful discussions, and also the anonymous reviewers of INDOCRYPT 2017 for helpful comments.

References

- [CGPZ16] Coron, J.-S., Greuet, A., Prouff, E., Zeitoun, R.: Faster evaluation of sboxes via common shares. In: Gierlichs and Poschmann [GP16], pp. 498–514
- [CJRR99] Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener [Wie99], pp. 398–412
- [Cor14] Coron, J.-S.: Higher order masking of look-up tables. In: Nguyen and Oswald [NO14], pp. 441–458
- [CPR07] Coron, J.-S., Prouff, E., Rivain, M.: Side channel cryptanalysis of a higher order masking scheme. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 28–44. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_3
- [CRZ17] Coron, J.-S., Rondepierre, F., Zeitoun, R.: High order masking of look-up tables with common shares. IACR Cryptology ePrint Archive, 2017:271 (2017)
- [DDF14] Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: Nguyen and Oswald [NO14], pp. 423–440
- [GP16] Gierlichs, B., Poschmann, A.Y. (eds.): CHES 2016. LNCS, vol. 9813. Springer, Heidelberg (2016). <https://doi.org/10.1007/978-3-662-53140-2>
- [GR16] Goudarzi, D., Rivain, M.: On the multiplicative complexity of boolean functions and bitsliced higher-order masking. In: Gierlichs and Poschmann [GP16], pp. 457–478

- [GR17] Goudarzi, D., Rivain, M.: How fast can higher-order masking be in software? In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 567–597. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_20
- [GRVV17] Goudarzi, D., Rivain, M., Vergnaud, D., Vivek, S.: Generalized polynomial decomposition for s-boxes with application to side-channel countermeasures. IACR Cryptology ePrint Archive 2017:632 (2017)
- [ISW03] Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
- [JS17] Journault, A., Standaert, F.-X.: Very high order masking: Efficient implementation and security evaluation. IACR Cryptology ePrint Archive 2017:637 (2017)
- [KJJ99] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener [Wie99], pp. 388–397
- [Koc96] Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
- [NO14] Nguyen, P.Q., Oswald, E. (eds.): EUROCRYPT 2014. LNCS, vol. 8441. Springer, Heidelberg (2014). <https://doi.org/10.1007/978-3-642-55220-5>
- [PR07] Prouff, E., Rivain, M.: A generic method for secure SBox implementation. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 227–244. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77535-5_17
- [PR13] Prouff, E., Rivain, M.: Masking against side-channel attacks: a formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_9
- [PRB09] Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Trans. Comput. **58**(6), 799–811 (2009)
- [PV16] Pulkus, J., Vivek, S.: Reducing the number of non-linear multiplications in masking schemes. In: Gierlichs and Poschmann [GP16], pp. 479–497
- [RDP08] Rivain, M., Dottax, E., Prouff, E.: Block ciphers implementations provably secure against second order side channel analysis. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 127–143. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_8
- [RRST02] Rao, J.R., Rohatgi, P., Scherzer, H., Tinguely, S.: Partitioning attacks: Or how to rapidly clone some GSM cards. In: 2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, 12–15 May 2002, pp. 31–41. IEEE Computer Society (2002)
- [SP06] Schramm, K., Paar, C.: Higher order masking of the AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 208–225. Springer, Heidelberg (2006). https://doi.org/10.1007/11605805_14
- [SVO+10] Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: another look on second-order DPA. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 112–129. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_7

- [Vad17] Vadnala, P.K.: Time-memory trade-offs for side-channel resistant implementations of block ciphers. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 115–130. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_7
- [Wie99] Wiener, M. (ed.): CRYPTO 1999. LNCS, vol. 1666. Springer, Heidelberg (1999). <https://doi.org/10.1007/3-540-48405-1>

Several Masked Implementations of the Boyar-Peralta AES S-Box

Ashrudit Ghoshal¹(✉)  and Thomas De Cnudde² 

¹ Indian Institute of Technology Kharagpur, Kharagpur, India
ashruditg@iitkgp.ac.in

² ESAT-COSIC and imec, KU Leuven, Leuven, Belgium
thomas.decnudde@esat.kuleuven.be

Abstract. Threshold implementation is a masking technique that provides provable security for implementations of cryptographic algorithms against power analysis attacks. In recent publications, several different threshold implementations of AES have been designed. However in most of the threshold implementations of AES, the Canright S-Box has been used. The Boyar-Peralta S-Box is an alternative implementation of the AES S-Box with a minimal circuit depth and is comparable in size to the frequently used Canright AES S-Box. In this paper, we present several versions of first-order threshold implementations of the Boyar-Peralta AES S-Box with different number of shares and several trade-offs in area, randomness and speed. To the best of our knowledge these are the first threshold implementations of the Boyar-Peralta S-Box. Our implementations compare favourably with some of the existing threshold implementations of Canright S-Box along the design trade-offs, e.g. while one of our S-Boxes is 49% larger in area than the smallest known threshold implementation of the Canright AES S-Box, it uses 63% less randomness and requires only 50% of the clock cycles. We provide results of a practical security evaluation based on real power traces to confirm the first-order attack resistance of our implementations.

Keywords: AES · Boyar-Peralta S-box · Countermeasure · DPA
Masking · SCA · Threshold Implementations

1 Introduction

In a black box model, embedded devices have been shown to be secure using modern ciphers. However, when naively implemented, side-channel information like power consumption, electromagnetic radiations or timing of the device's computations can leak secret information unintentionally. Attacks based on various side channels were presented in [16, 23, 24] and their mitigation has been the subject of a great deal of research ever since.

Masking is an efficient way to strengthen cryptographic implementations against such physical side-channel attacks [10, 18]. Masking detaches leaked side-channel information from secret dependent intermediate values by carrying out

computations on randomized values. It offers provable security [29] and can be implemented on the algorithmic level, making it a flexible Side-Channel Analysis (SCA) countermeasure. The underlying principle of masking relies on splitting each variable into a set of random values using secret sharing techniques and using a certain multi-party computation protocol on the resulting random values for secure computations. Once the secret values are masked, they are in no way combined until the end of the algorithm, i.e. the sensitive values are not leaked at any point during the execution of the cryptographic algorithm. Only at the end of the computation, when the cipher's outputs are valid, the output masks are combined to reconstruct the unmasked output.

The security of masking schemes is inherently tied to an adversary model. An attacker who observes the d^{th} -order statistical moment of e.g. a power trace or combines observations from d points in time nonlinearly in that power trace is said to be an attacker mounting a d^{th} -order attack. To prevent a d^{th} -order attack, a masking scheme of order $(d+1)$ is required. Fortunately, the number of readings needed for a higher-order attack to become successful grows exponentially with the noise standard deviation and therefore it is reasonable to guarantee practical security up to a certain order.

Implementing masking in hardware in a secure manner is not trivial. It is a delicate job since all the assumptions made on the leakage behavior of the underlying platform do not always hold in practice. For example, glitches are a known predominant threat [25] to the security of masked implementations in CMOS technologies. Some masking schemes like Threshold Implementations (TI) work under assumptions which are more achievable in a practical scenarios. In addition to these relaxed assumptions on the underlying leakage, TI offers provable security and allows to construct secure circuits which are realistic in size, all without requiring much intervention from a designer or many design iterations. For this reason, TI has been applied to many well-known cryptographic algorithms like KECCAK, AES and PRESENT [3, 14, 26, 28].

The Canright S-Box [9] and Boyar-Peralta S-Box [8] are two of the smallest implementations of the AES S-Box. As a starting point for threshold implementations and Side-Channel Analysis (SCA) secure designs, the Canright S-box has been used predominantly [5, 20, 26], whereas the Boyar-Peralta S-box has received little to no attention. The S-box introduced by Boyar and Peralta [8] is based on a novel logic minimization technique, which can be applied to any arbitrary combinational logic problems and even circuits that have been optimized by standard methodologies. The authors described their techniques as a two-step process: a reduction of nonlinear gates and a reduction of linear gates. Using their method they came up with an S-Box for AES which has the smallest combinational circuit depth known till date.

The aim of this paper is to develop secure masked implementations of the Boyar-Peralta AES S-Box using TI. The Boyar-Peralta S-Box is one of the smallest circuits implementing the AES S-box in unmasked form. We explore whether it is also one of the smallest masked S-Box of AES. For this purpose we explore

several different masking styles of the Boyar-Peralta S-Box, focusing on various trade-offs between area, randomness and the number of clock cycles.

Contributions. We present the first threshold implementations of the Boyar-Peralta AES S-Box. More precisely, we show TIs of the Boyar-Peralta AES S-Box with 3 and 4 shares, both with various trade-offs related to the circuit area, the consumed randomness and the required clock cycles. We consider two approaches to mask the S-Box. The first approach involves masking the AND gates alone using uniform sharing of the individual AND gates. The second approach is based on sharing a larger algebraic function, the $\mathbb{GF}(2^4)$ inverter as a whole.

Our smallest implementation is 2.75% larger in area than the smallest Canright S-Box presented in [6] but reduces randomness required by 37.5% and takes the same number of clock cycles. This implementation of ours which is the smallest in area takes as many clock cycles as the fastest known Threshold Implementation of the Canright S-Box. The Canright S-Box in [15] is the smallest known TI of the AES S-Box so far. Our smallest implementation is 47% larger in area but reduces randomness by 63% and increases speed by 50%. One of our implementations uses no randomness at all while all known threshold implementations of the Canright S-Box need randomness. We show the results of leakage detection tests of our implementations on a low noise FPGA platform to back up the theoretical security.

Organization. In Sect. 2, we provide the notation and the theory behind the threshold implementations masking scheme and the Boyar-Peralta AES S-Box. In Sect. 3, we develop the various secure implementations of the Boyar-Peralta S-Box by successively reducing either the number of shares, or the required randomness when the number of shares is kept constant. We present the results of the side-channel analysis in Sect. 4. In Sect. 5, we discuss the implementation cost of our resulting designs and compare them with costs of related previously published threshold implementations. We conclude the paper and propose directions for future work in Sect. 6.

2 Preliminaries

2.1 Notation

We use lowercase regular and bold letters to describe elements of $\mathbb{GF}(2^n)$ and their sharing respectively. Any sensitive variable $x \in \mathbb{GF}(2^n)$ is split into s shares $(x_1, \dots, x_s) = \mathbf{x}$, where $x_i \in \mathbb{GF}(2^n)$, in the initialization phase of the cryptographic algorithm. A possible manner of performing this initialization, which we employ, is as follows: the shares x_1, x_2, \dots, x_{s-1} are selected randomly from an uniform distribution and x_s is calculated such that $x = \bigoplus_{i \in \{1, 2, \dots, s\}} x_i$. We refer to the j^{th} bit of x as x^j unless $x \in \mathbb{GF}(2)$. We use the same notation to share a function f into s shares $\mathbf{f} = (f_1, \dots, f_s)$. The number of input and output shares of \mathbf{f} are denoted by s_{in} and s_{out} respectively. We refer to field multiplication as \times , to addition as \oplus and denote negation of all bits in a value x using \bar{x} .

2.2 The Boyar-Peralta Implementation of the AES S-Box

The Boyar-Peralta S-Box, is a circuit of depth 16 introduced by Boyar and Peralta [8]. It uses a total of 128 2-input gates to construct the S-Box: 94 gates are linear operations (XOR and XNOR gates) and 34 gates are nonlinear (AND gates or 1-bit multiplications).

The circuit is divided into 3 layers:

1. the top linear layer
2. the middle nonlinear layer
3. the bottom linear layer

The equations involved are listed below. The 8 input bits are given by $u_0, u_1, u_2, u_3, u_4, u_5, u_6$ and u_7 with u_0 being the most significant bit and u_7 being the least significant bit. Similarly, the 8 output bits are given by $s_0, s_1, s_2, s_3, s_4, s_5, s_6$ and s_7 , with s_0 being the most significant bit and s_7 being the least significant bit.

The set of equations for the top linear layer are:

$$\begin{array}{lll}
 t_1 = u_0 \oplus u_3 & t_{10} = t_6 \oplus t_7 & t_{19} = t_7 \oplus t_{18} \\
 t_2 = u_0 \oplus u_5 & t_{11} = u_1 \oplus u_5 & t_{20} = t_1 \oplus t_{19} \\
 t_3 = u_0 \oplus u_6 & t_{12} = u_2 \oplus u_5 & t_{21} = u_6 \oplus u_7 \\
 t_4 = u_3 \oplus u_5 & t_{13} = t_3 \oplus t_4 & t_{22} = t_7 \oplus t_{21} \\
 t_5 = u_4 \oplus u_6 & t_{14} = t_6 \oplus t_{11} & t_{23} = t_2 \oplus t_{22} \\
 t_6 = t_1 \oplus t_5 & t_{15} = t_5 \oplus t_{11} & t_{24} = t_2 \oplus t_{10} \\
 t_7 = u_1 \oplus u_2 & t_{16} = t_5 \oplus t_{12} & t_{25} = t_{20} \oplus t_{17} \\
 t_8 = u_7 \oplus t_6 & t_{17} = t_9 \oplus t_{16} & t_{26} = t_3 \oplus t_{16} \\
 t_9 = u_7 \oplus t_7 & t_{18} = u_3 \oplus u_7 & t_{27} = t_1 \oplus t_{12}
 \end{array}$$

The set of equations for the middle nonlinear layer are given by:

$$\begin{array}{lll}
 m_1 = t_{13} \times t_6 & m_{17} = m_5 \oplus t_{24} & m_{33} = m_{27} \oplus m_{25} \\
 m_2 = t_{23} \times t_8 & m_{18} = m_8 \oplus m_7 & m_{34} = m_{21} \times m_{22} \\
 m_3 = t_{14} \oplus m_1 & m_{19} = m_{10} \oplus m_{15} & m_{35} = m_{24} \times m_{34} \\
 m_4 = t_{19} \times u_7 & m_{20} = m_{16} \oplus m_{13} & m_{36} = m_{24} \oplus m_{25} \\
 m_5 = m_4 \oplus m_1 & m_{21} = m_{17} \oplus m_{15} & m_{37} = m_{21} \oplus m_{29} \\
 m_6 = t_3 \times t_{16} & m_{22} = m_{18} \oplus m_{13} & m_{38} = m_{32} \oplus m_{33} \\
 m_7 = t_{22} \times t_9 & m_{23} = m_{19} \oplus t_{25} & m_{39} = m_{23} \oplus m_{30} \\
 m_8 = t_{26} \oplus m_6 & m_{24} = m_{22} \oplus m_{23} & m_{40} = m_{35} \oplus m_{36} \\
 m_9 = t_{20} \times t_{17} & m_{25} = m_{22} \times m_{20} & m_{41} = m_{38} \oplus m_{40} \\
 m_{10} = m_9 \oplus m_6 & m_{26} = m_{21} \oplus m_{25} & m_{42} = m_{37} \oplus m_{39} \\
 m_{11} = t_1 \times t_{15} & m_{27} = m_{20} \oplus m_{21} & m_{43} = m_{37} \oplus m_{38} \\
 m_{12} = t_4 \times t_{27} & m_{28} = m_{23} \oplus m_{25} & m_{44} = m_{39} \oplus m_{40} \\
 m_{13} = m_{12} \oplus m_{11} & m_{29} = m_{28} \times m_{27} & m_{45} = m_{42} \oplus m_{41} \\
 m_{14} = t_2 \times t_{10} & m_{30} = m_{26} \times m_{24} & m_{46} = m_{44} \times t_6 \\
 m_{15} = m_{14} \oplus m_{11} & m_{31} = m_{20} \times m_{23} & m_{47} = m_{40} \times t_8 \\
 m_{16} = m_3 \oplus m_2 & m_{32} = m_{27} \times m_{31} & m_{48} = m_{39} \times u_7
 \end{array}$$

$$\begin{array}{lll}
 m_{49} = m_{43} \times t_{16} & m_{54} = m_{41} \times t_{10} & m_{59} = m_{38} \times t_{22} \\
 m_{50} = m_{38} \times t_9 & m_{55} = m_{44} \times t_{13} & m_{60} = m_{37} \times t_{20} \\
 m_{51} = m_{37} \times t_{17} & m_{56} = m_{40} \times t_{23} & m_{61} = m_{42} \times t_1 \\
 m_{52} = m_{42} \times t_{15} & m_{57} = m_{39} \times t_{19} & m_{62} = m_{45} \times t_4 \\
 m_{53} = m_{45} \times t_{27} & m_{58} = m_{43} \times t_3 & m_{63} = m_{41} \times t_2
 \end{array}$$

The set of equations for the bottom linear layer consist of:

$$\begin{array}{lll}
 l_0 = m_{61} \oplus m_{62} & l_{13} = m_{50} \oplus l_0 & l_{26} = l_7 \oplus l_9 \\
 l_1 = m_{50} \oplus m_{56} & l_{14} = m_{52} \oplus m_{61} & l_{27} = l_8 \oplus l_{10} \\
 l_2 = m_{46} \oplus m_{48} & l_{15} = m_{55} \oplus l_1 & l_{28} = l_{11} \oplus l_{14} \\
 l_3 = m_{47} \oplus m_{55} & l_{16} = m_{56} \oplus l_0 & l_{29} = l_{11} \oplus l_{17} \\
 l_4 = m_{54} \oplus m_{58} & l_{17} = m_{57} \oplus l_1 & s_0 = l_6 \oplus l_{24} \\
 l_5 = m_{49} \oplus m_{61} & l_{18} = m_{58} \oplus l_8 & s_1 = \overline{l_{16} \oplus l_{26}} \\
 l_6 = m_{62} \oplus l_5 & l_{19} = m_{63} \oplus l_4 & s_2 = \overline{l_{19} \oplus l_{28}} \\
 l_7 = m_{46} \oplus l_3 & l_{20} = l_0 \oplus l_1 & s_3 = l_6 \oplus l_{21} \\
 l_8 = m_{51} \oplus m_{59} & l_{21} = l_1 \oplus l_7 & s_4 = l_{20} \oplus l_{22} \\
 l_9 = m_{52} \oplus m_{53} & l_{22} = l_3 \oplus l_{12} & s_5 = \overline{l_{25} \oplus l_{29}} \\
 l_{10} = m_{53} \oplus l_4 & l_{23} = l_{18} \oplus l_2 & s_6 = \overline{l_{13} \oplus l_{27}} \\
 l_{11} = m_{60} \oplus l_2 & l_{24} = l_{15} \oplus l_9 & s_7 = \overline{l_6 \oplus l_{23}} \\
 l_{12} = m_{48} \oplus m_{51} & l_{25} = l_6 \oplus l_{10} &
 \end{array}$$

Masked software implementations using bitslicing of the Boyar Peralta AES S-Box were studied in [19, 22]. A modified version of the Boyar Peralta S-Box has been masked using the ISW AND gate [21] in [19].

2.3 Threshold Implementations

The threshold implementations (TI) masking technique was proposed by Nikova et al. [27] as a countermeasure against Differential Power Analysis (DPA) attacks. It is secure even in non-ideal circuits where glitches have shown to result in leakage in more conventional masking schemes [25]. The original proposal, which only dealt with first-order DPA security, was later extended to protect against higher-order DPA attacks as well [4, 30].

TI is based on multi-party computation and secret sharing, and must satisfy the following properties in order to achieve the mentioned security:

1. **Uniformity.** Uniformity requires all intermediate shares to be uniformly distributed. It ensures state-independence from the mean of the leakages, which is a requirement to thwart first-order DPA. As mentioned in [2] it suffices to check uniformity at the inputs and the outputs of each of the functions. Uniformity can be either achieved through correction terms by using more input shares, or by adding randomness after the non-uniform computation.
2. **Non-completeness.** To achieve d^{th} -order non-completeness, any combination of d or less component functions f_i of \mathbf{f} must be independent of at least one input share x_i . For protection against first-order DPA, 1st-order non-completeness is required, i.e. every function must be independent of at least

one input share. Non-completeness ensures that the side-channel security of the final circuit is not affected by glitches.

3. **Correctness.** This property simply states that applying the sub-functions to a valid shared input must always yield a valid sharing of the correct output.

In addition to TI's algorithmic properties, the physical leakage of each share or sub-function should be independent of all other shares or sub-functions, i.e. no coupling is present between the shares or sub-functions. Violating this assumption has shown to induce leakage in masked implementations [13].

3 Several SCA Secure Implementations of the Boyar-Peralta AES S-Box

In this section we present several different threshold implementations of the Boyar-Peralta AES S-Box. Applying TI to linear functions is straightforward due to the linearity of the XOR and XNOR operations. Masking the nonlinear functions on the other hand is known to pose more of a challenge. As mentioned in the previous section the only nonlinear functions in the Boyar-Peralta AES S-Box are the AND gates. In order to apply TI to these AND gates we need to make sure the resulting sharings are non-complete and correct, and that their outputs are uniform. In our first approach, we therefore consider the uniform sharing of an AND gate and formulate several 1st-order non-complete TI sharings for this S-box. We additionally investigate a second approach: instead of masking each AND gate individually, we combine several AND gates to form an inversion in $\mathbb{GF}(2^4)$. In both cases, to avoid first-order leakages from glitches and early propagation of signals, each masked nonlinear function must be followed by a set of registers.

The middle layer is the nonlinear layer in the Boyar-Peralta AES S-Box. The top and the bottom layer are composed of linear functions only. When we mask each gate individually, the outputs of every AND gate in the middle layer must be registered before the next operation starts. Hence, we divide the middle layer into stages such that at each stage, the outputs produced by the AND gates are put into registers before proceeding for the operation in the next stage.

On inspection of the set of equations, we divide the circuit into 4 stages where each stage ends with a set of AND operations. Note that there may be other ways to divide the nonlinear layer into stages. The top linear layer was combined with the first stage of the nonlinear layer and the outputs of the AND gates from the 4th and final stage of the middle nonlinear layer are fed into the bottom layer directly, which causes no problem since this layer is linear. Therefore, we divide our circuit into 4 stages with a set of registers after the first three stages. A total of 4 clock cycles are required to complete the computation of the S-Box. The entire circuit of the nonlinear middle layer is shown in Fig. 1. The set of equations after division into stages are given below.

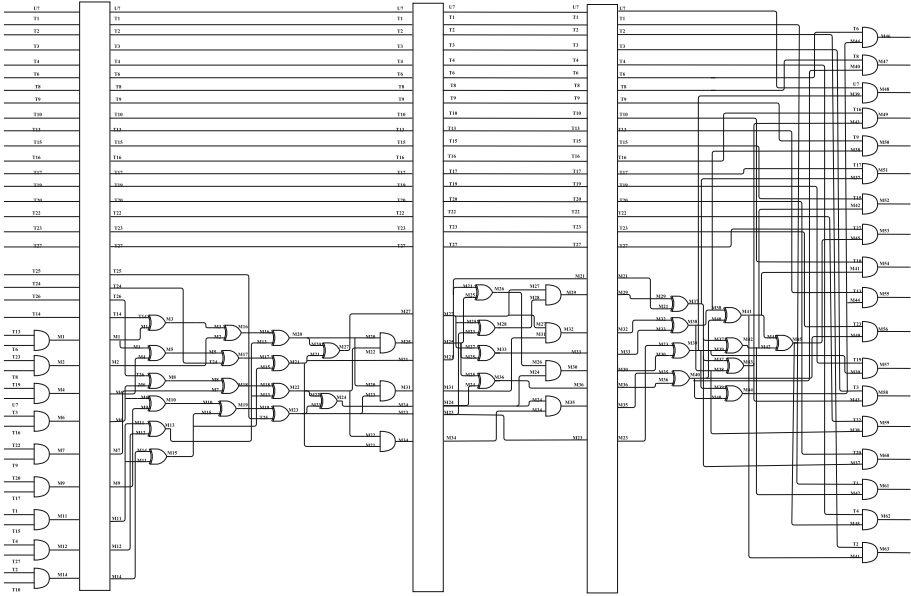


Fig. 1. Division of the nonlinear layer into stages.

Stage 1

$t_1 = u_0 \oplus u_3$	$t_{13} = t_3 \oplus t_4$	$t_{25} = t_{20} \oplus t_{17}$
$t_2 = u_0 \oplus u_5$	$t_{14} = t_6 \oplus t_{11}$	$t_{26} = t_3 \oplus t_{16}$
$t_3 = u_0 \oplus u_6$	$t_{15} = t_5 \oplus t_{11}$	$t_{27} = t_1 \oplus t_{12}$
$t_4 = u_3 \oplus u_5$	$t_{16} = t_5 \oplus t_{12}$	$m_1 = t_{13} \times t_6$
$t_5 = u_4 \oplus u_6$	$t_{17} = t_9 \oplus t_{16}$	$m_2 = t_{23} \times t_8$
$t_6 = t_1 \oplus t_5$	$t_{18} = u_3 \oplus u_7$	$m_4 = t_{19} \times u_7$
$t_7 = u_1 \oplus u_2$	$t_{19} = t_7 \oplus t_{18}$	$m_6 = t_3 \times t_{16}$
$t_8 = u_7 \oplus t_6$	$t_{20} = t_1 \oplus t_{19}$	$m_7 = t_{22} \times t_9$
$t_9 = u_7 \oplus t_7$	$t_{21} = u_6 \oplus u_7$	$m_9 = t_{20} \times t_{17}$
$t_{10} = t_6 \oplus t_7$	$t_{22} = t_7 \oplus t_{21}$	$m_{11} = t_1 \times t_{15}$
$t_{11} = u_1 \oplus u_5$	$t_{23} = t_2 \oplus t_{22}$	$m_{12} = t_4 \times t_{27}$
$t_{12} = u_2 \oplus u_5$	$t_{24} = t_2 \oplus t_{10}$	$m_{14} = t_2 \times t_{10}$

Stage 2

$m_3 = t_{14} \oplus m_1$	$m_{17} = m_5 \oplus t_{24}$	$m_{24} = m_{22} \oplus m_{23}$
$m_5 = m_4 \oplus m_1$	$m_{18} = m_8 \oplus m_7$	$m_{25} = m_{22} \times m_{20}$
$m_8 = t_{26} \oplus m_6$	$m_{19} = m_{10} \oplus m_{15}$	$m_{27} = m_{20} \oplus m_{21}$
$m_{10} = m_9 \oplus m_6$	$m_{20} = m_{16} \oplus m_{13}$	$m_{31} = m_{20} \times m_{23}$
$m_{13} = m_{12} \oplus m_{11}$	$m_{21} = m_{17} \oplus m_{15}$	$m_{34} = m_{21} \times m_{22}$
$m_{15} = m_{14} \oplus m_{11}$	$m_{22} = m_{18} \oplus m_{13}$	
$m_{16} = m_3 \oplus m_2$	$m_{23} = m_{19} \oplus t_{25}$	

Stage 3

$$\begin{aligned} m_{26} &= m_{21} \oplus m_{25} \\ m_{28} &= m_{23} \oplus m_{25} \\ m_{29} &= m_{28} \times m_{27} \end{aligned}$$

$$\begin{aligned} m_{30} &= m_{26} \times m_{24} \\ m_{32} &= m_{27} \times m_{31} \\ m_{33} &= m_{27} \oplus m_{25} \end{aligned}$$

$$\begin{aligned} m_{35} &= m_{24} \times m_{34} \\ m_{36} &= m_{24} \oplus m_{25} \end{aligned}$$

Stage 4

$$\begin{aligned} m_{37} &= m_{21} \oplus m_{29} \\ m_{38} &= m_{32} \oplus m_{33} \\ m_{39} &= m_{23} \oplus m_{30} \\ m_{40} &= m_{35} \oplus m_{36} \\ m_{41} &= m_{38} \oplus m_{40} \\ m_{42} &= m_{37} \oplus m_{39} \\ m_{43} &= m_{37} \oplus m_{38} \\ m_{44} &= m_{39} \oplus m_{40} \\ m_{45} &= m_{42} \oplus m_{41} \\ m_{46} &= m_{44} \times t_6 \\ m_{47} &= m_{40} \times t_8 \\ m_{48} &= m_{39} \times u_7 \\ m_{49} &= m_{43} \times t_{16} \\ m_{50} &= m_{38} \times t_9 \\ m_{51} &= m_{37} \times t_{17} \\ m_{52} &= m_{42} \times t_{15} \\ m_{53} &= m_{45} \times t_{27} \\ m_{54} &= m_{41} \times t_{10} \\ m_{55} &= m_{44} \times t_{13} \\ m_{56} &= m_{40} \times t_{23} \\ m_{57} &= m_{39} \times t_{19} \\ m_{58} &= m_{43} \times t_3 \end{aligned}$$

$$\begin{aligned} m_{59} &= m_{38} \times t_{22} \\ m_{60} &= m_{37} \times t_{20} \\ m_{61} &= m_{42} \times t_1 \\ m_{62} &= m_{45} \times t_4 \\ m_{63} &= m_{41} \times t_2 \\ l_0 &= m_{61} \oplus m_{62} \\ l_1 &= m_{50} \oplus m_{56} \\ l_2 &= m_{46} \oplus m_{48} \\ l_3 &= m_{47} \oplus m_{55} \\ l_4 &= m_{54} \oplus m_{58} \\ l_5 &= m_{49} \oplus m_{61} \\ l_6 &= m_{62} \oplus l_5 \\ l_7 &= m_{46} \oplus l_3 \\ l_8 &= m_{51} \oplus m_{59} \\ l_9 &= m_{52} \oplus m_{53} \\ l_{10} &= m_{53} \oplus l_4 \\ l_{11} &= m_{60} \oplus l_2 \\ l_{12} &= m_{48} \oplus m_{51} \\ l_{13} &= m_{50} \oplus l_0 \\ l_{14} &= m_{52} \oplus m_{61} \\ l_{15} &= m_{55} \oplus l_1 \\ l_{16} &= m_{56} \oplus l_0 \end{aligned}$$

$$\begin{aligned} l_{17} &= m_{57} \oplus l_1 \\ l_{18} &= m_{58} \oplus l_8 \\ l_{19} &= m_{63} \oplus l_4 \\ l_{20} &= l_0 \oplus l_1 \\ l_{21} &= l_1 \oplus l_7 \\ l_{22} &= l_3 \oplus l_{12} \\ l_{23} &= l_{18} \oplus l_2 \\ l_{24} &= l_{15} \oplus l_9 \\ l_{25} &= l_6 \oplus l_{10} \\ l_{26} &= l_7 \oplus l_9 \\ l_{27} &= l_8 \oplus l_{10} \\ l_{28} &= l_{11} \oplus l_{14} \\ l_{29} &= l_{11} \oplus l_{17} \\ s_0 &= l_6 \oplus l_{24} \\ s_1 &= l_{16} \oplus l_{26} \\ s_2 &= l_{19} \oplus l_{28} \\ s_3 &= l_6 \oplus l_{21} \\ s_4 &= l_{20} \oplus l_{22} \\ s_5 &= l_{25} \oplus l_{29} \\ s_6 &= l_{13} \oplus l_{27} \\ s_7 &= l_6 \oplus l_{23} \end{aligned}$$

For the second approach, where we mask the circuit using the inversion in $\mathbb{GF}(2^4)$ instead of masking each individual AND gate. $m_{20}m_{21}m_{22}m_{23}$ are inputs to the $\mathbb{GF}(2^4)$ inverter and $m_{36}m_{32}m_{39}m_{28}$ being the output where m_{20} and m_{36} are the most significant bits of the input and output respectively. $m_{20}, m_{21}, m_{22}, m_{23}$ become available in Stage 2. The part of the circuit in Stage 2 to obtain $m_{20}, m_{21}, m_{22}, m_{23}$ is linear. Hence we can put the inverter right after $m_{20}, m_{21}, m_{22}, m_{23}$ become available without using a register. The outputs of the inverter $m_{36}, m_{32}, m_{39}, m_{28}$ were the outputs of Stage 3. Therefore, we combine Stage 2 and 3 to isolate the inverter. The modified set of equations are given below:

Stage 1

$$\begin{array}{lll}
t_1 = u_0 \oplus u_3 & t_{16} = t_7 \oplus t_{15} & m_4 = t_{19} \times u_7 \\
t_2 = u_0 \oplus u_5 & t_{17} = t_9 \oplus t_{16} & m_5 = m_4 \oplus m_1 \\
t_3 = u_0 \oplus u_6 & t_{19} = t_9 \oplus u_3 & m_6 = t_3 \times t_{16} \\
t_4 = u_3 \oplus u_5 & t_{20} = t_1 \oplus t_{19} & m_7 = t_{22} \times t_9 \\
t_{13} = t_3 \oplus t_4 & t_{22} = t_9 \oplus u_6 & m_8 = m_7 \oplus m_6 \\
t_5 = u_4 \oplus t_{13} & t_{23} = t_2 \oplus t_{22} & m_9 = t_{20} \times t_{17} \\
t_6 = t_5 \oplus u_5 & t_{24} = t_2 \oplus t_{10} & m_{10} = m_9 \oplus m_6 \\
t_7 = u_1 \oplus u_2 & t_{25} = t_{20} \oplus t_{17} & m_{11} = t_1 \times t_{15} \\
t_8 = u_7 \oplus t_6 & t_{26} = t_3 \oplus t_{16} & m_{12} = t_4 \times t_{27} \\
t_9 = u_7 \oplus t_7 & t_{27} = t_{10} \oplus t_{15} & m_{13} = m_{12} \oplus m_{11} \\
t_{10} = t_6 \oplus t_7 & m_1 = t_{13} \times t_6 & m_{14} = t_2 \times t_{10} \\
t_{14} = t_5 \oplus u_1 & m_2 = t_{23} \times t_8 & \\
t_{15} = t_{14} \oplus t_1 & m_3 = m_2 \oplus m_1 &
\end{array}$$

Stage 2

$$\begin{array}{lll}
m_{15} = m_{14} \oplus m_{11} & m_{18} = m_8 \oplus m_{13} & m_{21} = m_{17} \oplus t_{24} \\
m_{16} = m_3 \oplus m_{13} & m_{19} = m_{10} \oplus m_{15} & m_{22} = m_{18} \oplus t_{26} \\
m_{17} = m_5 \oplus m_{15} & m_{20} = m_{16} \oplus t_{14} & m_{23} = m_{19} \oplus t_{25}
\end{array}$$

$m_{20}m_{21}m_{22}m_{23}$ are inputs to the $\mathbb{GF}(2^4)$ inverter and $m_{36}m_{32}m_{39}m_{28}$ being the output where m_{20} and m_{36} are the most significant bits of the input and output respectively.

Stage 3

$$\begin{array}{lll}
m_{40} = m_{39} \oplus m_{36} & z_{12} = m_{42} \times t_3 & l_{12} = z_3 \oplus z_5 \\
m_{41} = m_{28} \oplus m_{32} & z_{13} = m_{39} \times t_{22} & l_{13} = z_{13} \oplus l_1 \\
m_{42} = m_{28} \oplus m_{39} & z_{14} = m_{28} \times t_{20} & l_{14} = l_4 \oplus l_{12} \\
m_{43} = m_{32} \oplus m_{36} & z_{15} = m_{41} \times t_1 & s_3 = l_3 \oplus l_{11} \\
m_{44} = m_{40} \oplus m_{41} & z_{16} = m_{44} \times t_4 & l_{16} = z_6 \oplus l_8 \\
z_0 = m_{43} \times t_6 & z_{17} = m_{40} \times t_2 & l_{17} = z_{14} \oplus l_{10} \\
z_1 = m_{36} \times t_8 & l_1 = z_{15} \oplus z_{16} & l_{18} = \overline{l_{13} \oplus l_{14}} \\
z_2 = m_{32} \times u_7 & l_2 = z_{10} \oplus l_1 & s_7 = z_{12} \oplus l_{18} \\
z_3 = m_{42} \times t_{16} & l_3 = z_9 \oplus l_2 & l_{20} = z_{15} \oplus l_{16} \\
z_4 = m_{39} \times t_9 & l_4 = z_0 \oplus z_2 & l_{21} = l_2 \oplus z_{11} \\
z_5 = m_{28} \times t_{17} & l_5 = z_1 \oplus z_0 & s_0 = \overline{l_3 \oplus l_{16}} \\
z_6 = m_{41} \times t_{15} & l_6 = z_3 \oplus z_4 & s_6 = \overline{l_{10} \oplus l_{18}} \\
z_7 = m_{44} \times t_{27} & l_7 = z_{12} \oplus l_4 & s_4 = \overline{l_{14} \oplus s_3} \\
z_8 = m_{40} \times t_{10} & l_8 = z_7 \oplus l_6 & s_1 = s_3 \oplus l_{16} \\
z_9 = m_{43} \times t_{13} & l_9 = z_8 \oplus l_7 & l_{26} = \overline{l_{17} \oplus l_{20}} \\
z_{10} = m_{36} \times t_{23} & l_{10} = l_8 \oplus l_9 & s_2 = \overline{l_{26} \oplus z_{17}} \\
z_{11} = m_{32} \times t_{19} & l_{11} = l_6 \oplus l_5 & s_5 = l_{21} \oplus l_{17}
\end{array}$$

The circuit of the middle nonlinear layer using an inverter is shown in Fig. 2.

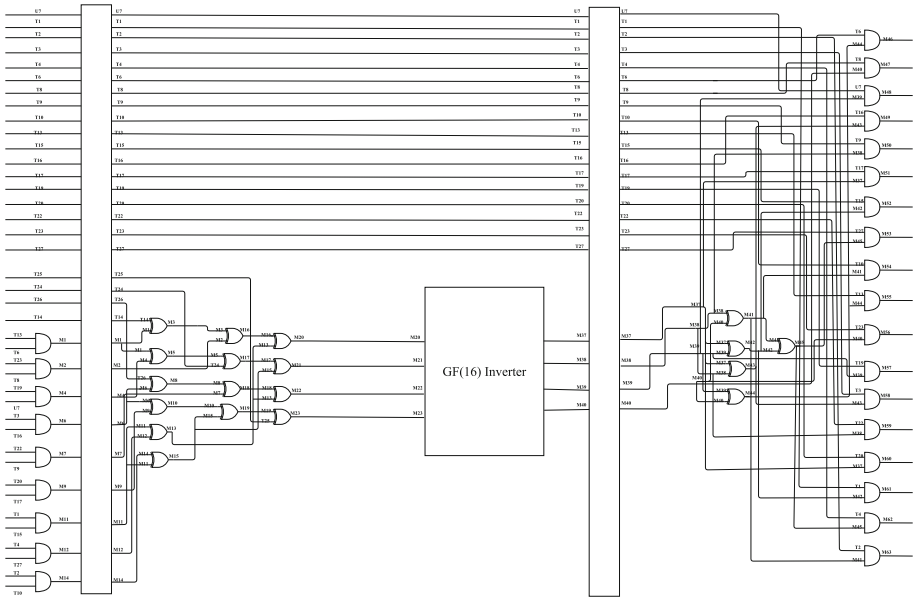


Fig. 2. Division of the nonlinear layer into stages when centered around the inversion in $\mathbb{GF}(2^4)$.

Using these two different approaches for division into stages of the circuit, we design the following secure implementations of the Boyar-Peralta S-Box:

1. Threshold implementation with 4 shares and no randomness in Sect. 3.1
2. Threshold implementation with 3 shares and 68 bits randomness in Sect. 3.2
3. Threshold implementation with 3 shares and 34 bits of randomness in Sect. 3.3
4. Threshold Implementation using 3 shares and using sharing with $s_{in} = 5$ and $s_{out} = 5$ for a $\mathbb{GF}(2^4)$ inverter in Sect. 3.4
5. Threshold Implementation using 3 shares and using sharing with $s_{in} = 4$ and $s_{out} = 4$ for a $\mathbb{GF}(2^4)$ inverter in Sect. 3.5

3.1 Threshold Implementation with 4 Shares and No Randomness

As previously mentioned, the sharing for the linear operations is trivial. For the nonlinear AND gate we first use the following uniform 4-to-4 sharing. This is a novel modification of a 4-to-3 uniform sharing of the AND gate used in [5].

$$\begin{aligned}
\mathbf{a} &= \mathbf{x} \times \mathbf{y} \\
\mathbf{x} &= (x_1, x_2, x_3, x_4) \\
\mathbf{y} &= (y_1, y_2, y_3, y_4) \\
\mathbf{a} &= (a_1, a_2, a_3, a_4) \\
a_1 &= (x_2 \oplus x_3 \oplus x_4) \times (y_2 \oplus y_3) \oplus y_3 \\
a_2 &= ((x_1 \oplus x_3) \times (y_1 \oplus y_4)) \oplus (x_1 \times y_3) \oplus x_4 \\
a_3 &= (x_2 \oplus x_4) \times (y_1 \oplus y_4) \oplus x_4 \oplus y_4 \\
a_4 &= (x_1 \times y_2) \oplus y_3
\end{aligned}$$

The complete computation of the S-Box will take 4 clock cycles and will not consume any randomness.

3.2 Threshold Implementation with 3 Shares and 68 Bits Randomness

Having designed a threshold implementation for the Boyar-Peralta AES S-Box which uses no randomness, we now aim to reduce the size of our circuit. This can be achieved by reducing the number of shares.

There is however no 3-to-3 uniform sharing for a 2-input AND gate. To keep the uniformity of sharing property intact, we introduce some randomness to remask the shares as shown in [26]. We use the following 3-to-3 sharing of the 2 input AND gate. r_1, r_2 are the 2 bits of randomness.

$$\begin{aligned}
\mathbf{a} &= \mathbf{x} \times \mathbf{y} \\
\mathbf{x} &= (x_1, x_2, x_3) \\
\mathbf{y} &= (y_1, y_2, y_3) \\
\mathbf{a} &= (a_1, a_2, a_3) \\
a_1 &= (x_2 \times y_2) \oplus (x_2 \times y_3) \oplus (x_3 \times y_2) \oplus r_1 \oplus r_2 \\
a_2 &= (x_3 \times y_3) \oplus (x_1 \times y_3) \oplus (x_3 \times y_1) \oplus r_2 \\
a_3 &= (x_1 \times y_1) \oplus (x_1 \times y_2) \oplus (x_2 \times y_1) \oplus r_1
\end{aligned}$$

One masked AND gate consumes 2-bits of randomness. The whole S-Box circuit requires $2 \times 34 = 68$ bits of randomness in total. The complete computation of the S-Box will take 4 clock cycles.

3.3 Threshold Implementation with 3 Shares and 34 Bits Randomness

We now reduce the amount of randomness required in our circuit by using the technique of virtual sharing as used in [7]. This sharing uses 1 bit of randomness

per 2-input AND gate. The following is the resulting 3-to-3 sharing of the 2-input AND gate using 1 bit of randomness. r denotes a bit of randomness.

$$\begin{aligned}
 \mathbf{a} &= \mathbf{x} \times \mathbf{y} \\
 \mathbf{x} &= (x_1, x_2, x_3) \\
 \mathbf{y} &= (y_1, y_2, y_3) \\
 \mathbf{a} &= (a_1, a_2, a_3) \\
 a_1 &= (x_2 \times y_2) \oplus (x_2 \times y_3) \oplus (x_3 \times y_2) \oplus r \\
 a_2 &= (x_3 \times y_3) \oplus (x_1 \times y_3) \oplus (x_3 \times y_1) \oplus (x_1 \times r) \oplus (y_1 \times r) \\
 a_3 &= (x_1 \times y_1) \oplus (x_1 \times y_2) \oplus (x_2 \times y_1) \oplus (x_1 \times r) \oplus (y_1 \times r) \oplus r
 \end{aligned}$$

This S-Box circuit requires 34 bits of randomness. The complete computation of the S-Box will again take 4 clock cycles.

3.4 Threshold Implementation Using 3 Shares and Using Sharing with $s_{in} = 5$ and $s_{out} = 5$ for a $\mathbb{GF}(2^4)$ Inverter

As stated earlier, we can isolate an inverter in $\mathbb{GF}(2^4)$ within the Boyar-Peralta S-Box. As shown in [5] we can use a 5-to-5 uniform sharing for this $\mathbb{GF}(2^4)$ inverter. We use 3 shares for the linear and nonlinear gates that fall outside the inverter. In order to increase the number of shares from 3 to 5 at the input of the inverter we use 4 extra bits of randomness. To reduce the number of shares at the output from 5 to 3 we use 2 bits of randomness to combine the output shares just after the register. As mentioned in [2] uniformity is necessary only for the input of nonlinear functions. The part of the circuit before the inverter in Stage 2 is linear. In order increase in the number of shares before input to the inverter, the shares are remasked using randomness. Therefore before input to the nonlinear part of Stage 2, the inverter, the shares are uniform due to remasking. Hence inputs to stage 2 i.e. outputs of Stage 1 need not be uniform. Also all the outputs of the AND operations in stage 3 are inputs linear functions, hence they need not be uniform. This version has 27 2-input AND gates. All of them are in stages 1 and 3. Since the outputs of Stages 1 and 3 need not be uniform, none of the AND gates need to be uniform. So, we may use any non-complete and correct 3 sharing without using randomness for these AND gates. The total amount of randomness required is $4 \times 4 \times 4 + 2 \times 4 = 24$ bits.

3.5 Threshold Implementation Using 3 Shares and Using Sharing with $s_{in} = 4$ and $s_{out} = 4$ for a $\mathbb{GF}(2^4)$ inverter

Similar to the previous implementation, we again use the threshold implementation of the $\mathbb{GF}(2^4)$ inverter. There is 4-to-4 sharing of the $\mathbb{GF}(2^4)$ inverter which is not uniform. We observe that for decreasing the output shares from 4 to 3, we add randomness to the outputs, which essentially remasks the outputs and provides uniformity.

The circuit differs from the previous one only in the aspects that the shares are increased from 3 to 4 and decreased from 4 to 3, and that the sharing for the inverter itself is different. It takes the same number of clock cycles as the previous one, i.e. 3, but requires 3 bits of randomness for increasing the number of shares from 3 to 4 and then 2 bits of randomness for reducing the shares back from 4 to 3. The argument to not use a uniform sharing of AND gates used in the previous implementation is applicable here too. Hence a total of $3 \times 4 + 2 \times 4 = 20$ bits of randomness is required.

4 Side-Channel Analysis Evaluation

First, we describe the circuit that we used for the sequential evaluation of the S-Boxes. All the S-Boxes have separate input ports for the input shares and the randomness, and separate output ports for the output shares. Each S-Box has an enable signal and a reset signal as input. The execution of the S-Box begins when the enable signal is set to high. The values at the ports having the input shares and randomness for the corresponding S-Box, at the time enable goes high, are the ones used as the input to the S-Box. The reset signal is used to reset the S-Box to a known state. Each S-Box has an output done signal which goes high after the execution of the S-Box is complete and the outputs at the corresponding ports of output shares are the results of the execution of the S-Box.

There is an outer wrapper encapsulating the 5 S-Boxes. The wrapper has a control module. The control module of the wrapper has an enable as input signal and a complete signal as an output signal. The enable signal is needed to start the sequence of S-Boxes. The start signal of the first S-Box is set to high on the positive edge of clock following the enable signal going high. When the done signal of the S-Box goes high, the control waits for a few clock cycles before setting the start signal of the next S-Box high. After the done signal of the last S-Box goes high, the complete signal of the wrapper is set to high. The

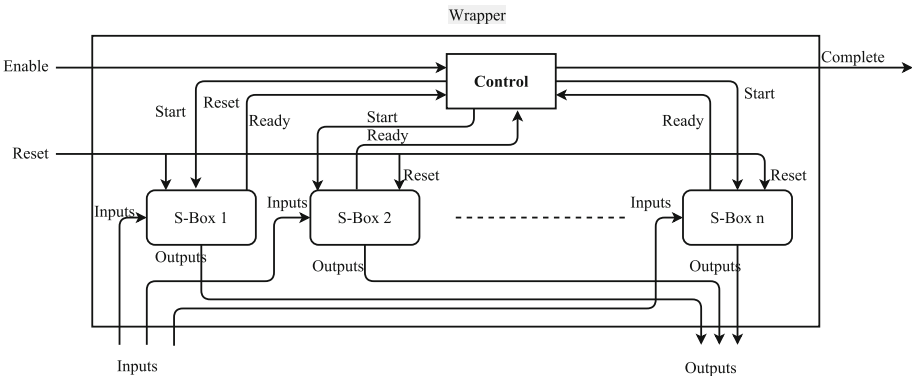


Fig. 3. Structure of circuit for sequential evaluation of the S-Boxes

wrapper has a reset signal as an input which is sent as the reset signal of the S-Boxes when set to high resets all S-Boxes to known states. Figure 3 shows the structure of the wrapper.

The design was implemented on a SASEBO-G measurement board using Xilinx ISE 10.1 in order to analyze their leakage characteristics.

The SASEGO-G board has two Xilinx Virtex-II Pro FPGA devices. Our design, was implemented on the crypto FPGA(xc2vp7). In order to prevent optimizations over module boundaries, the “Keep Hierarchy” constraint was kept on while generating the programming file. The control FPGA (xc2vp30) is responsible for the I/O with the measurement PC and generation of random bits. The PRNG which the control FPGA uses to generate the input sharings and random masks for the S-boxes is an AES-128 in OFB mode.

We evaluate the security of our first order secure implementations of the Boyar Peralta AES S-Boxes. We use leakage detection tests [1, 11, 12, 17, 24] to test for any power leakage of our masked implementations. The fix class of the leakage detection is chosen as the zero plaintext in all our evaluations.

We follow the standard practice when testing a masked design i.e. first turn off the PRNG to switch off the masking countermeasure. The design is expected to show leakage in this setting, and this serves to confirm that the experimental setup is sound (we can detect leakage). We then proceed by turning on the PRNG. If we do not detect leakage in this setting, the masking countermeasure is deemed to be effective. Figures 4, 5, 6, 7 and 8 show the result of the first order leakage detection tests on the S-Boxes.

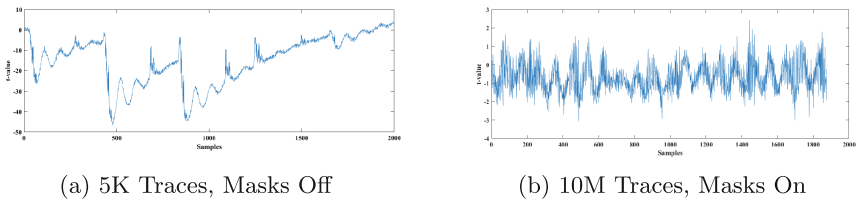


Fig. 4. First Order leakage detection test for the S-Box with 4 shares.

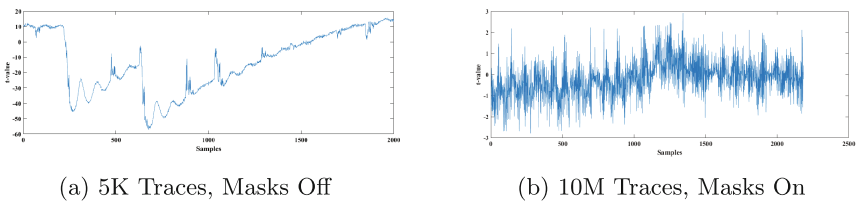


Fig. 5. First Order leakage detection test for the S-Box with 3 shares, 68 bits of randomness

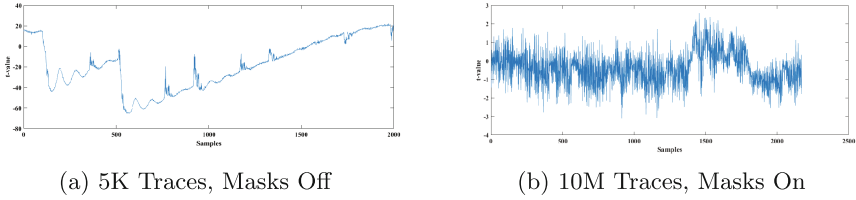


Fig. 6. First Order leakage detection test for the S-Box with 3 shares, 34 bits of randomness

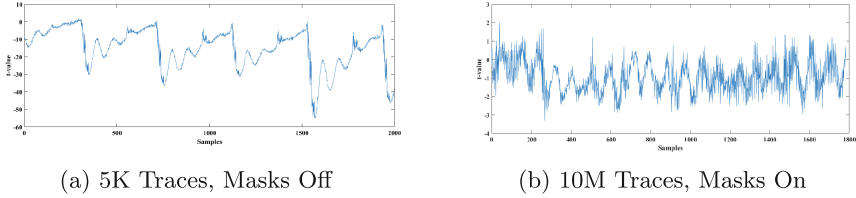


Fig. 7. First Order leakage detection test for the S-Box with 3 shares and using sharing with $s_{in} = 5$ and $s_{out} = 5$ for a $\mathbb{GF}(2^4)$ inverter

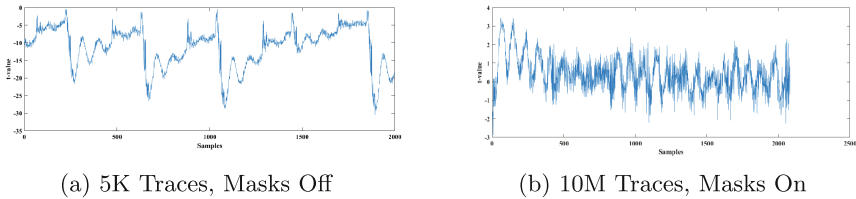


Fig. 8. First Order leakage detection test for the S-Box with 3 shares and using sharing with $s_{in} = 4$ and $s_{out} = 4$ for a $\mathbb{GF}(2^4)$ inverter

5 Implementation Cost

Here we give a comparison of the area, the required randomness and the number of clock cycles for our implementations. The results of area have been obtained using Synopsys 2013.12 and NanGate 45 nm Open Cell Library.

In Table 1, we observe a trade-off between randomness, area and clock cycles. As we reduce the area, the randomness per S-box lookup increases or the number of clock cycles required increase. In our implementation with smallest area, where we share a large algebraic function, the $\mathbb{GF}(2^4)$ inverter as a whole, both the number of clock cycles and the area are reduced.

In Table 2 we compare our implementation with smallest area to some masked implementations based on Canright S-Box.

Table 1. Area, Randomness and Clock cycles required per S-box implementation.

	Area [GEs]	Randomness [bits]	Clock cycles
Unprotected	269	0	1
$s_{in} = 4, s_{out} = 4$	4609	0	4
$s_{in} = 3, s_{out} = 3$, 68 random bits, individual AND gates masked	3630	68	4
$s_{in} = 3, s_{out} = 3$, 34 random bits, individual AND gates masked	3798	34	4
$s_{in} = 3, s_{out} = 3$, inverter masked with $s_{in} = 5, s_{out} = 5$	3344	24	3
$s_{in} = 3, s_{out} = 3$, inverter masked with $s_{in} = 4, s_{out} = 4$	2913	20	3

Table 2. Area, Randomness and Clock Cycles required per S-box for related Implementations.

	Area [GEs]	Randomness [bits]	Clock cycles
$s_{in} = 3, s_{out} = 3$, inverter masked with $s_{in} = 4, s_{out} = 4$, Boyar Peralta	2914	20	3
$s_{in} = 3, s_{out} = 3$, Canright S-Box in [5]	3708	44	3
$s_{in} = 3, s_{out} = 3$, Canright S-Box in [26]	4244	48	4
$s_{in} = 3, s_{out} = 3$, Canright S-Box in [6]	2835	32	3
$s_{in} = 2, s_{out} = 2$, Canright S-Box in [15]	1977	54	6

We can summarize the comparison of our implementations with related implementations as follows:

- We achieve an implementation that consumes no randomness.
- Two of our implementations, which use the sharing for inversion in $\mathbb{GF}(2^4)$, take 3 clock cycles, which is faster than implementations in [15,26]
- Our implementation that uses the 4-sharing of an inverter needs the same number of clock cycles as the smallest one in [6], while consuming less randomness for an increase in area of only 2.75%.
- The S-Box in [15] is the smallest known TI of the AES S-Box. Our implementation is 47% larger in comparison but we obtain a 63% reduction in randomness of and 50% reduction in number of clock cycles required.

6 Conclusion

In this paper, we present the first threshold implementations of the Boyar-Peralta AES S-Box. Since this AES S-Box is of minimum known depth, the critical path might be smaller which would allow clocking the core at higher frequencies making it highly important for secure high-speed and high-throughput applications. We go through an iterated design process, starting from a straightforward approach where we mask each gate individually to arrive at a more efficient implementation by masking the larger algebraic structure of the inversion in $\mathbb{GF}(2^4)$.

Our smallest implementation is 49% larger in area compared to the smallest known threshold implementation of the Canright AES S-box but reduces the randomness by 63% and number of clock cycles by 50%. Moreover, we achieve a secure implementation of the AES S-Box that requires no randomness at all. The set of secure implementations we present gives the hardware designer more options for tailoring their implementations according to their specifications.

A future direction of research can investigate the result of starting from a masked Canright AES S-Box and using the optimizations mentioned in [8] to arrive at a small and secure implementation of the Boyar-Peralta S-Box. Masking the Boyar Peralta S-Box with $d+1$ shares as shown in [30] is a possible direction for future work. Another future work would be designing circuits for this S-Box with higher-order security levels, as a determined adversary can still break the first-order masking scheme with a second order attack.

Acknowledgements. We would like to thank Prof. Vincent Rijmen for his valuable comments and feedback on the paper and the anonymous reviewers for providing constructive and valuable comments. This work was supported in part by NIST with the research grant 60NANB15D34, in part by the Research Council KU Leuven, OT/13/071 and by the Flemish Government through FWO project Cryptography secured against side-channel attacks by tailored implementations enabled by future technologies (G0842.13). Ashrujit Ghoshal was a Visiting Scholar at ESAT-COSIC, KU Leuven hosted by Prof. Vincent Rijmen. Thomas De Cnudde is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

References

1. Becker, G., Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Kouzminov, T., Leiserson, A., Marson, M., Rohatgi, P., et al.: Test vector leakage assessment (TVLA) methodology in practice. In: International Cryptographic Module Conference, vol. 1001, p. 13 (2013)
2. Bilgin, B.: Threshold implementations: as countermeasure against higher-order differential power analysis (2015)
3. Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., Van Assche, G.: Efficient and first-order DPA resistant implementations of KECCAK. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 187–199. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_13

4. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_18
5. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: A more efficient AES threshold implementation. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT 2014. LNCS, vol. 8469, pp. 267–284. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06734-6_17
6. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-offs for threshold implementations illustrated on AES. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* **34**(7), 1188–1200 (2015)
7. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Stütz, G.: Threshold implementations of all 3×3 and 4×4 S-boxes. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 76–91. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_5
8. Boyar, J., Peralta, R.: A small depth-16 circuit for the AES S-box. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IAICT, vol. 376, pp. 287–298. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30436-1_24
9. Canright, D.: A very compact S-box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_32
10. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26
11. Coron, J.S., Naccache, D., Kocher, P.: Statistics and secret leakage. *ACM Trans. Embed. Comput. Syst. (TECS)* **3**(3), 492–508 (2004)
12. Coron, J.-S., Kocher, P., Naccache, D.: Statistics and secret leakage. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 157–173. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45472-1_12
13. De Cnudde, T., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., Rijmen, V.: Does coupling affect the security of masked implementations? In: Guilley, S. (ed.) COSADE 2017. LNCS, vol. 10348, pp. 1–18. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64647-3_1
14. De Cnudde, T., Bilgin, B., Reparaz, O., Nikov, V., Nikova, S.: Higher-order threshold implementation of the AES S-box. In: Homma, N., Medwed, M. (eds.) CARDIS 2015. LNCS, vol. 9514, pp. 259–272. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31271-2_16
15. De Cnudde, T., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with $d+1$ shares in hardware. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 194–212. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_10
16. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_21
17. Gilbert Goodwill, B.J., Jaffe, J., Rohatgi, P., et al.: A testing methodology for side-channel resistance validation. In: NIST Non-invasive Attack Testing Workshop (2011)
18. Goubin, L., Patarin, J.: DES and differential power analysis the “Duplication” method. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_15

19. Goudarzi, D., Rivain, M.: How fast can higher-order masking be in software? In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 567–597. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_20
20. Gross, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. IACR Cryptology ePrint Archive 2016/486 (2016)
21. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
22. Journault, A., Standaert, F.-X.: Very high order masking: efficient implementation and security evaluation. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 623–643. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_30
23. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
24. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
25. Mangard, S., Pramstaller, N., Oswald, E.: Successfully attacking masked AES hardware implementations. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 157–171. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_12
26. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: a very compact and a threshold implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_6
27. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 529–545. Springer, Heidelberg (2006). https://doi.org/10.1007/11935308_38
28. Poschmann, A., Moradi, A., Khoo, K., Lim, C.-W., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2, 300 GE. *J. Cryptol.* **24**(2), 322–345 (2011)
29. Prouff, E., Rivain, M.: Masking against side-channel attacks: a formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_9
30. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_37

Author Index

- Adomnicai, Alexandre 65
Al Musa, Saud 206
Angel, Jiji 147
Ashokkumar, C. 147
- Berger, Thierry P. 247
Beullens, Ward 227
Bonnoron, Guillaume 27
Boyle, Elette 1
Buchmann, Johannes A. 346
- Celik, Turku Ozlum 44
Chatterjee, Sanjit 85
Chattopadhyay, Anupam 282
- De Cnudde, Thomas 384
Dunkelman, Orr 166
Duquesne, Sylvain 186
- Fontaine, Caroline 27
Fournier, Jacques J. A. 65
- Gaborit, Philippe 247
Gallin, Gabriel 44
Ghammam, Loubna 186
Ghoshal, Ashrujit 384
- Haber, Simi 166
Holzer, Patrick 346
Huang, Tao 302
- Khairallah, Mustafa 282
Khandaker, Md. Al-Amin 186
Kodera, Yuta 186
- Lallemand, Virginie 126
- Masson, Laurent 65
Menezes, Bernard 147
Mukherjee, Sayantan 85
Mukhopadhyay, Debdeep 267
- Nanjo, Yuki 186
Nogami, Yasuyuki 186
- Pandit, Tapas 85
Paterson, Kenneth G. 107
Peyrin, Thomas 282
Preneel, Bart 227
- Rahul, R. 147
Rasoolzadeh, Shahram 126
Rosulek, Mike 325
Ruatta, Olivier 247
- Sadhukhan, Rajat 267
Sarkar, Sumanta 267
Syed, Habeeb 267
- Tisserand, Arnaud 44
Tjuawinata, Ivan 302
- Villanueva-Polanco, Ricardo 107
Vivek, Srinivas 369
- Weizmann, Ariel 166
Wu, Hongjun 302
Wunderer, Thomas 346
- Xu, Guangwu 206